UNIVERSITY OF CALIFORNIA

Los Angeles

# Theory, Design and Characterization of Protein Symmetry Combination Materials

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy in Molecular Biology

by

Joshua Laniado

2020

## ABSTRACT OF THE DISSERTATION

Theory, Design and Characterization of Protein Symmetry Combination Materials

by

Joshua Laniado

Doctor of Philosophy in Molecular Biology

University of California, Los Angeles, 2020

Professor Todd O. Yeates, Chair

Nature has evolved a plethora of sophisticated protein complexes to carry out fundamental biological processes. While most of these exquisite macromolecular machines exhibit complex architectures, many are composed of only a few different types of subunits. Understanding how protein molecules combine to form these remarkable self-assembling structures only makes sense in the light of symmetry. By limiting the number of distinct interactions required between individual subunits, symmetry offers a simpler route for the evolution of supramolecular assemblies such as viral capsids and bacterial microcompartments.

Principles of symmetry and self-assembly have invigorated recent efforts in molecular engineering giving rise to a growing suite of novel protein materials such as finite cages and extended crystalline arrays. These designed assemblies are rapidly finding applications in areas as diverse as vaccine design, atomic imaging, enzyme scaffolding and molecular delivery. Despite significant advances in computational approaches and design strategies, constructing these materials remains extremely challenging. Here, we address key experimental and theoretical limitations to improve the prospects for the routine design of novel symmetric protein materials.

In Chapter 1, we review current methodologies for designing self-assembling protein nanomaterials. A first approach presented the idea that when two separate symmetric oligomers associate in some geometrically defined way, a structure with higher symmetry can be obtained through self-assembly. There, an alpha-helical linker is used to connect two oligomeric components and to control their relative geometry. A second approach does not involve genetic fusion but relies instead on the computational design of a novel protein-protein interface. After reviewing the successful constructions resulting from both methods, challenges and limitations are discussed. In the fusion approach, the inherent flexibility of the alpha helical linker can lead to the formation of unintended assemblies. Alternatively, the interface design strategy exhibits limited success in predicting viable protein interfaces. The prevalence of such limitations dramatically hinders the creation of novel materials, motivating the development of alternate strategies.

In the next chapter, we introduce a new approach for the design of symmetric self-assembling nanomaterials. Building upon the fusion approach, the original alpha-helical linker is replaced with a heterodimeric coiled coil as an attempt to reduce flexibility. Further, the use of a known heterodimeric interface to combine component oligomers alleviates the challenges associated with *de novo* interface design. Ten symmetric protein cages were designed using this method among which two were structurally characterized. One design assembled as intended while the other crystallized in an alternate form. Geometric distinctions between the two help explain the different degrees of success, leading to crucial lessons and establishing clearer principles for the creation of novel nanoscale protein architectures.

While some experimental aspects have been addressed, only a small fraction of the possible design space has been explored. That space, which is anticipated to offer a multitude of symmetry-based combinations, has not been described in theory. In Chapter 3, we articulate all of the possible kinds of protein-based materials that can be created by combining two symmetric oligomers. Specifically, 13 types of cages, 35 types of 2-D layers and 76 types of 3-D crystals are identified as possible targets for design. We lay out a complete rule set for constructing all such symmetry combination materials (SCMs) and introduce a unified system for parameterizing and searching the construction space for each case. This theoretical and computational study provides a blueprint for a blossoming area of macromolecular design.

Owing to the complexity and our limited understanding of the rules that govern protein behavior, designing protein-protein interfaces remains challenging. Current approaches rely on empirical or knowledge-based energy functions and optimization algorithms that often fail to produce stable interfaces. On the other hand, there is growing evidence that the database of known protein structures is now sufficiently large to cover the structural landscape of protein interfaces. In Chapter 4, we argue that carefully-selected structural motifs can be used as templates for interface design. We introduce Nanohedra, a fragment-based docking tool that harnesses the power of our theoretical framework to enable the design of all possible SCMs. Prospective designs of symmetric materials are proposed along with a retrospective analysis of recent design studies. In this analysis, our tool recapitulates all successful designs while poorly ranking failed ones. With a user-friendly interface and a unified protocol for symmetric protein design, Nanohedra enables the creation of a universe of novel nanomaterials and opens new avenues for nanobiotechnology.

The dissertation of Joshua Laniado is approved.

Z. Hong Zhou

Jose A. Rodriguez

Joseph A. Loo

James U. Bowie

Todd O. Yeates, Committee Chair

University of California, Los Angeles

2020

*For my mother and father who have made it all possible*

## Table of Contents

# List of Figures

## List of Tables

## Acknowledgments

Through possibly the longest, yet most impactful six years of my life, this run has been a compilation of ethereal joy and melancholic struggles. If one thing is unequivocal, I wouldn't be who I am today, if it were not for the richness of my experiences and the profound impact of those around me. The ability to take this on alone would be an impossible feat and I am immensely grateful to have had the utmost support of such wonderful people.

I would first like to extend my immeasurable gratitude to Professor Todd Yeates for his unwavering support and mentorship. Professor Yeates has never failed to fill me with inspiration. He has persistently compelled me to further my growth in science and in life. Todd, thank you for believing in me even at times when self-belief was hard to come by. Thank you for allowing me to seek advice whenever it felt necessary and for sharing your unparalleled knowledge in the fields of crystallography and computational biophysics. Most importantly, thank you for bestowing upon me your love for symmetry.

I want to thank my committee members, Professors Z. Hong Zhou, Jose A. Rodriguez, Joseph A. Loo and James U. Bowie for their invaluable insight, counsel and direction which have helped shape the way I think about science. I also would like to thank Professor Pascal Egea who has always so generously left his laboratory door open for me. His instrumental teachings have given me invaluable insight in the fields of molecular biology and protein biochemistry.

A distinct thank you to Drs. Michael Sawaya and Duilio Cascio. It has been an inconceivable privilege to learn from them and to work alongside them. Their exceptional teachings and unrivaled expertise in the field of structural biology have helped me expand my knowledge and formulate new ideas.

Thank you to Dr. Mark Arbing at the UCLA Protein Expression Technology Center for his advice and suggestions regarding protein expression and purification, as well as Michael Collazo for numerous insightful discussions and important technical assistance with protein crystallization. Thank you to the staff members of the UCLA Electron Imaging Center for NanoMachines, especially Ivo Atanasov and Wong Hoi Hui, who have been extremely resourceful specifically through their guidance on electron microscopy. Thank you to Ashley Terhorst and Cindy Chau for their constant support in administrative matters throughout my graduate studies.

I am profoundly grateful for all of the current and former members of the Yeates lab, especially Dr. Julien Jorda, Kyle Meador, Justin Miller, Dr. Yuxi Liu, Dr. Kevin Cannon, Dr. Dan McNamara, Jessica Ochoa and our lab manager Inna Pashkov. I thank them for their friendship, never-ending support, insightful advice and constructive feedback. It has been an utter pleasure and such a privilege to work alongside such brilliant minds.

I would like to extend my sincere gratitude to my friend Professor Raymond F. Schinazi for his perpetual guidance and motivation. Professor Schinazi continuously inspires me and drives me to apply my research and knowledge to real-world matters that require solutions.

I want to thank my siblings, Anna, Alon, and Jade for their love and support. I am thankful for my incredible friends, Saïd, Max, Clement, Jacques, Tina, Cam, Mimi, Aditya, Preston, Jordane, Kelly, and Matthias. Their unconditional support and patience throughout this journey mean the world to me. I am exceptionally grateful for my loving partner Jasmin. I thank her for her unworldly compassion, unconditional understanding and unwavering love. I would also like to thank my dogs Heems and Milo for their love and companionship. They have kept me grounded and they have never failed to bring me back to the present moment.

Last but not least, I would like to thank my mother Linda and my father Arye from the bottom of my heart. They have stood by me through all of my endeavors and have ceaselessly encouraged me to pursue my passions. They have always done their utmost to ensure that my dreams become reality. Thank you to my father for his relentless belief in me. He has always inspired me and given me the confidence to chase even the wildest of my wildest dreams. Thank you to my mother for her endless support, boundless care and unconditional love. She has always put her own priorities aside to help me with mine. I could never thank my parents enough for their selflessness, generosity and unfathomable love. I can only hope that one day I am able to do for my own children half of what my parents have done for me.

# ***Vita***

## Education

2011-2014, B.S. in Biochemistry, University College London

## Publications

Yeates TO, Liu Y, **Laniado J**. The design of symmetric protein nanomaterials comes of age in theory and practice. Curr Opin Struct Biol. 2016 Aug;39:134-143. doi: 10.1016/j.sbi.2016.07.003.

**Laniado J**, Yeates TO. A complete rule set for designing symmetry combination materials from protein molecules. Proc Natl Acad Sci U S A. 2020 Nov 25:202015183. doi: 10.1073/pnas.2015183117.

# CHAPTER 1

# The design of symmetric protein nanomaterials comes of age in theory and practice

Available online at www.sciencedirect.com

ScienceDirect

Current Opinion in Structural Biology

# The design of symmetric protein nanomaterials comes of age in theory and practice

Todd O Yeates[1,2,3], Yuxi Liu[1] and Joshua Laniado[3]

In nature, protein molecules have evolved as building blocks for the assembly of diverse and complex structures, many of which exhibit a high degree of symmetry. This observation has motivated a number of recent engineering efforts in which the advantages of symmetry have been exploited to design novel self-assembling protein structures of great size. Materials ranging from cages to extended two and three-dimensional arrays have been demonstrated. Especially for extended arrays, a vast number of geometrically different design types are possible. A table of geometric rules is provided for designing a universe of novel materials by combining two component symmetries.

**Addresses**
[1] UCLA Department of Chemistry and Biochemistry, United States
[2] UCLA-DOE Institute for Genomics and Proteomics, United States
[3] UCLA-Molecular Biology Institute, United States

Corresponding author: Yeates, Todd O (yeates@mbi.ucla.edu)

**Current Opinion in Structural Biology** 2016, **39**:134–143

This review comes from a themed issue on **Engineering and design**

Edited by **Dan S Tawfik** and **Raghavan Varadarajan**

For a complete overview see the Issue and the Editorial

Available online 29th July 2016

**http://dx.doi.org/10.1016/j.sbi.2016.07.003**

0959-440X/© 2016 Elsevier Ltd. All rights reserved.

## Introduction

Building blocks that have self-complimentary interfaces can self-assemble into elaborate structures. Nature serves as a rich source of inspiring specimens. At the macromolecular scale, viral capsids are quintessential examples, but other equally extraordinary macromolecular assemblies abound in nature (reviewed in [1–3]). The beauty and functional utility of these assemblies have long-motivated engineering efforts to create comparable structures in the laboratory. Beginning in the 1980s Ned Seeman pioneered ideas for using DNA molecules as building blocks for nanostructures [4]. Over the years, those ideas and various strategic variations led to the creation of elaborate supramolecular architectures and design patterns built from nucleic acids (reviewed in [5]). In nature, protein molecules have been the choice for the evolution of large assemblies with diverse form and function. But the engineering path to following Nature's lead has been challenged by the complexity of the rules that govern protein folding and assembly. To overcome those challenges, special strategies are needed.

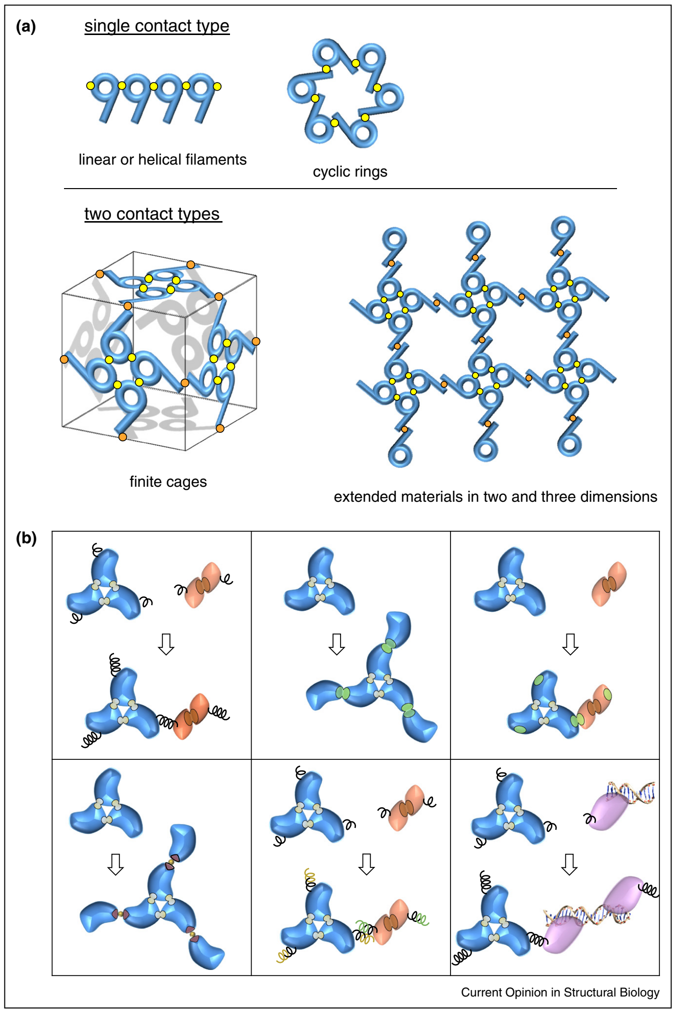
In developing a strategic approach for building with protein molecules, Nature provides a major clue. Symmetry prevails in naturally evolved protein assembles. This is an empirical fact evident in the vast database of known macromolecular structures [6,7], but the prevalence of high symmetry in large protein assemblies was anticipated at least as far back as 1956 when Crick and Watson emphasized that viral capsids were likely able to evolve more easily in symmetric forms because symmetric assemblies require the fewest number of distinct interfacial contacts between individual subunits [8]. That key observation applies as well to designed structures, and indeed the early history of designing protein assemblies is rich with cases of relatively simple symmetric structures such as dimers and helical filaments [9,10,11,12]. The push in recent years to create very large protein assemblies has been guided even more strongly by principles of symmetry.

## Symmetry-based design strategies

The symmetry of an object is fully described by the set of spatial operations (*e.g.* rotations) that leave the entire object unchanged except for an undetectable exchange of identical subunits. Because the symmetry of an object obeys the properties of a mathematical group, each specific type of symmetry is often referred to as a symmetry group. The symmetry group of a structure can be used to understand how many structurally distinct contact types are required to hold all the subunits together in one connected object. Certain simple types of architecture can be created from a building block that touches itself in just one way; i.e. using a single contact type. The possible outcomes are limited to structures like cyclic rings of subunits, or head-to-tail filaments (Figure 1a). More complex architectures require building blocks with more than one distinct interface.

A relatively simple group theory analysis explains the minimum number of distinct contact types required to achieve a given target symmetry. This was articulated first in the context of three-dimensional crystals [13] and then in the context of designed protein assemblies by Padilla *et al.* [14]. For example, if all the elements of a symmetry group can be generated by repeated application of a single element of the group (i.e. a rotational operation), then one contact type is sufficient. The cyclic or head-to-tail filament architectures noted above are examples of this type.

**Figure 1**


(a)
single contact type
linear or helical filaments
cyclic rings
two contact types
finite cages
extended materials in two and three dimensions
(b)
Current Opinion in Structural Biology

Assembly consequences and strategies for introducing multiple contact types into protein building blocks. **(a)** Illustration of varied symmetric architectural forms and the number of distinct contact types required for connectivity between molecular building blocks. Two contact types are sufficient to create diverse assemblies. **(b)** Different molecular strategies for creating a building block having two distinct contact types in a defined orientation. Left to right (top): alpha helical fusion; 1-component interface design; 2-component interface design. Left to right (bottom): metal or ligand bridging; coiled-coil helical fusions; designed symmetrization of DNA binding proteins.

If two elements from the symmetry group must be chosen in order to obtain the full symmetry group by repeated operations, then two contact types are required, and so on. Surprisingly, it turns out that a great many types of symmetry — including finite cages and many extended two and three-dimensional arrays — can be generated using just two properly chosen symmetry elements in combination (Figure 1a). This key point frames the problem of designing novel protein assemblies by prescribing the number of distinct contact types that must be built into a protein building block in order for it to assemble into the desired architecture. More than the minimum number of contact types can be present in a final assembly, but not fewer. The specific geometry of the interfaces is of course crucial for obtaining the desired result, and molecular strategies are required for creating these oriented interfaces.

Various strategies have been developed for building multiple distinct interfacial contact types into a protein molecule in order to generate elaborate supramolecular structures (Figure 1b). Padilla *et al.* [14] laid out a first strategy at a time when prospects for designing de novo interfaces into protein molecules were still remote. By necessity, naturally evolved interfaces were exploited by using simple natural protein oligomers (*e.g.* dimers and trimers) as a starting point. To create a single molecular building block containing two distinct interface types, a method was developed for genetically fusing two naturally oligomeric protein domains. In order to control relative geometry, only oligomeric domains having terminal alpha helices were considered, so that directly fusing two such proteins might create a geometrically predictable outcome if a continuous alpha helix was preserved between the two domains. The diversity of architectures possible by the general approach was described, and a first demonstration was provided — a 12-subunit assembly in the form of a tetrahedral cage was designed from a dimer plus trimer fusion [14]. This protein assembled into geometric structures consistent with the design, among a range of other polymorphic forms. Several years later, Lai *et al.* [15,16] showed that introducing two or three amino acid mutations into the originally designed protein sequence was sufficient to produce 12-subunit assemblies in high yield, which could be crystallized and validated in atomic detail. Not surprisingly, some flexibility of the helix linker gave rise to assemblies that were flexed or deformed from perfect symmetry, but which otherwise conformed to the intended tetrahedral design [16]. A different, 24-subunit cubic cage in good agreement with its design, and closely obeying the intended octahedral symmetry, was subsequently demonstrated using the same helix fusion strategy [17•].

Major leaps forward in design strategy were made by King *et al.* [18] working with globular protein domains and Lanci *et al.* [19] working with coiled-coil polypeptides; they foresaw that computational methods for introducing novel interfaces into protein surfaces by amino acid sequence design had matured to the point where they might allow large symmetric assemblies to be created. Grueninger *et al.*, took an earlier step in this direction by designing double-ring assemblies from naturally cyclic structures [20]. Following the symmetry ideas discussed above, starting with simple oligomeric proteins (*e.g.* dimers and trimers) means that only one additional interface needs to be designed into the protein in order to create complex architectures (Figure 1b). In King *et al.*, the procedure was enabled by a special algorithm written for the Rosetta-Design program to preserve overall symmetry while sampling the rigid body degrees of freedom available to the component oligomers [21]. From 41 initial designs, two cubic cages were produced in high yield and could be validated by crystallography [18]. One was a tetrahedral cage built from four trimers situated at the vertices of a tetrahedron and contacting each other primarily via a designed interface with two-fold symmetry. The other was a cubic/octahedral cage built from eight trimers at the corners of a cube, again with the trimers interacting primarily via a designed interface with two-fold symmetry. The polypeptide design work by Lanci *et al.* [19] relied on a trimeric coil-coil motif as the starting point; the introduction of lateral and vertical contacts gave a three dimensional crystalline material also validated by crystallography.

Within the bounds of symmetry-based methods of design, several strategic variations are possible beyond those noted above. Some of the possibilities are briefly described here (Figure 1b). Sinclair *et al.* [22] introduced a variation on the oligomer fusion method that relaxed the requirement for a continuous alpha helical linker; it applies to certain extended two and three-dimensional assemblies where two oligomers can be fused in a way that preserves a rotational symmetry element they both share. A few cases of well-ordered layers were demonstrated with that approach. Sciore *et al.* created a 24-subunit octahedral cage by fusing a naturally trimeric protein to a sequence that forms a parallel tetrameric coiled coil [42•]. The geometry of the connection in that case was also uncontrolled, but the limited range of possible outcomes from combining those two symmetries favored assembly to a single type of structure. King *et al.* [23••] introduced a two-component variation on the de novo interface design strategy. Two different natural oligomers comprise the starting materials, and computational sequence design is used to introduce a heterotypic interface between the two subunit types. By relying on two separate oligomeric components, the idea shares similarity with the helix fusion method. But the helix fusion is rendered unnecessary by the designed interface between the two oligomers. Furthermore, the non-covalent nature of the association between the two oligomeric components enables production and purification of separate components, with full assembly occurring upon mixing. In the first application of the two-component strategy, a series of approximately 60 designed tetrahedral cages were tested experimentally, and five were validated in detail by crystallography [23••,24]. A new application of the 2-component approach by Bale *et al.* has produced a whole suite of even larger cages with icosahedral symmetry, the largest of which is approximately 400 Å in diameter [25••].

In other design approaches, metals or bivalent ligands have been used as a way to introduce a new interface or self-associating interaction between oligomeric components ([26,27,28], reviewed in [29]). The addition of metals promotes assembly when metal binding half-sites (*e.g.* two suitably disposed histidine residues) are designed into a protein surface. Without further computational design of protein–protein interactions beyond the metal site, the metal site approach tends to give rise to assemblies whose outcomes are hard to predict in detail owing to alternate possible orientations of the metal ligands. Interestingly, despite incomplete control over geometry, various reports [30–35,36••,37•] have shown that the approach can be used to create diverse and intriguing materials ranging from small oligomers to helical structures to dynamic layers and three-dimensional arrays. A particularly notable variation was demonstrated recently by the introduction of a Zn-binding site at the three-fold symmetry axis of the natural, cubically symmetric ferritin cage. Adding a bivalent, metal-binding organic ligand joins the cubic protein cages into a three-dimensional body centered crystal lattice in a predictable fashion [36••]. In a report by Sakai *et al.*, the authors used bivalent organic ligands to create an additional dimeric subunit association for linking D2 tetramers together, in an unspecified orientation in this case. Changing the spacer length between the binding moieties on the organic ligand resulted in protein molecules held together in layers, which grew into two different types of three-dimensional crystals [38]. The design of rigid polyvalent ligands offers prospects for connecting proteins together in geometrically controlled ways. Particularly in view of the large number of proteins that naturally bind rigid (*e.g.* multi-ring) ligands, such a strategy remains relatively untapped.

Other strategic variations for combining symmetry elements are possible but have not been deeply explored (Figure 1b). DNA (or RNA) provides a facile route for directing molecular associations by complimentary base pairing. This can be exploited in combination with DNA binding proteins and either interface design or oligomeric fusion to create hybrid materials composed of proteins and nucleic acids. A strategy along that line has been taken by Mou *et al.* in creating linear or helical filaments [39]. In another study, Brodin *et al.* attached multiple copies of two different DNA molecules onto the surfaces of proteins to create two component types, which then assembled into cubically packed crystals upon addition of a complimentary bridging DNA molecule [40•]. DNA and RNA both offer an easy route for introducing a symmetry element into designed assemblies using palindromic sequences. The construction of symmetric, hybrid protein-nucleic acid materials using such a strategy has not been tested yet.

## Designed protein cages

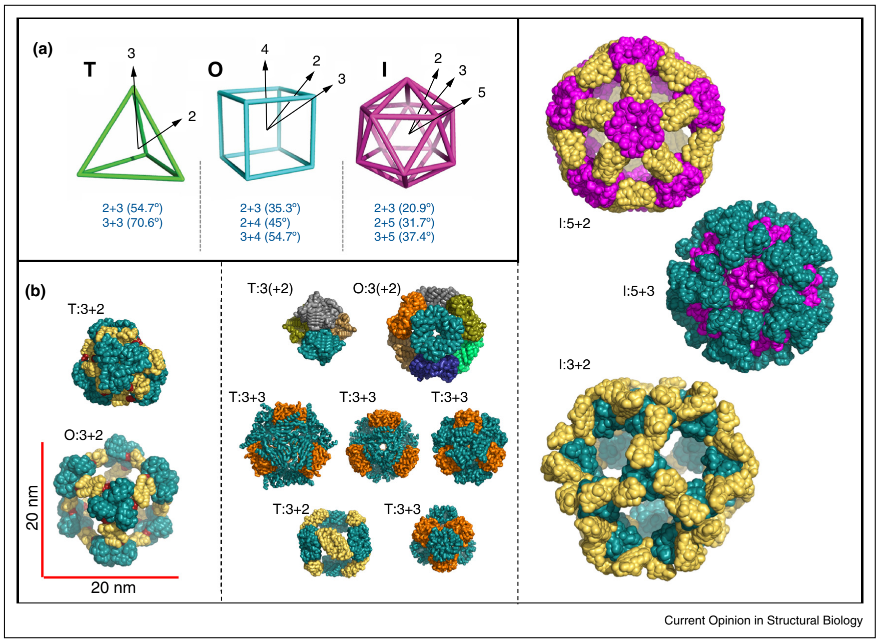
Designed assemblies of defined, finite size can take the form of shells or cages or more compact clusters following one of the three possible cubic symmetries in three dimensions. These follow from the symmetries of the five Platonic solids; the cube and the octahedron are duals of each other while the icosahedron and the dodecahedron are duals, so together with the tetrahedron (which is its own dual), there are three possible symmetries: T (order 12), O (order 24), and I (order 60). The symmetry rules for generating these have been articulated [14,41] (Figure 2a). For each symmetry type, nearly every combination of two *intersecting* symmetry axes is a possible choice for generating the final structure, but there are some exceptions (Figure 2 legend).

Using designed protein molecules, cages or shells belonging to all three of the cubic symmetry types (T, O, and I) have been successfully produced. Those that have been validated in atomic detail by X-ray crystallography are shown in Figure 2b. Each of the target symmetries has been obtained using multiple different symmetry combinations. Symmetry T has been obtained by combinations of two components or interfaces having 2-fold plus 3-fold symmetry [14,15,18,23••] and 3-fold plus 3-fold symmetry [23••]. Symmetry O has been obtained by 3-fold plus 2-fold symmetries [17•,18,23••] and by 4-fold plus 3-fold symmetries [42•]. Finally, icosahedral symmetry (I) has been obtained in new studies, using all three possible symmetry combinations (2 plus 3, 2 plus 5, and 3 plus 5) [25••,43•]. Together, these designed architectures range in number of subunits from 12 to 120, with diameters from 11 to 40 nm and masses from 276 kDa to 2.8 MDa. They cover most of the available symmetry options for creating cage and shell structures. The numerous possible applications for designed protein cages have been reviewed elsewhere [44–47] and are not elaborated here other than to summarize that practical uses are likely to include both interior and exterior capabilities: (i) encapsulation, delivery and release of molecular cargo [48–53], and (ii) polyvalent display of motifs for signaling or antigenic effects, as in synthetic vaccines [54,55].

## Design rules for building extended materials in two and three dimensions

In contrast to the finite cage designs that arise from two intersecting symmetry axes as discussed above, when two component symmetries are combined in an arrangement where any of their axes are not intersecting, the result cannot be finite and must instead be an extended or unbounded material. Filaments are one possible outcome, arising from two non-intersecting 2-fold axes of symmetry [14]. But more complex outcomes are obtained by combining higher symmetries. There, unbounded materials that extend as either 2-dimensional layers or 3-dimensional arrays (i.e. crystals) are possible. The geometric rules for a few possibilities of this type were laid out earlier on the basis of the combination of 2-fold and 3-fold axes of symmetry [14]. Beyond those, a vast range of possibilities arise from combinations of higher component symmetries. A few designs within that scope have

**Figure 2**



Design and validation of self-assembling protein cages with high symmetry. **(a)** The three types of cubic symmetry (T, O, and I) are illustrated on the framework of the Platonic solids. Angles between pairs of rotational symmetry axes that can be combined to create a self-assembling building block with the target symmetry are listed. **(b)** Engineered protein cages obeying all three possible cubic symmetries have been produced; a subset of structures that have been validated by X-ray crystallography are shown to scale. Left panel: a tetrahedral cage (PDB 4IQ4) [16] and an octahedral cage (PDB 4QCC) [17•] designed using the alpha helical fusion strategy [14]. Natural trimers are in blue; natural dimers are in yellow; the alpha-helical linkers are in red. Middle panel: tetrahedral cages and an octahedral cage engineered by de novo interface design with either one or two components. Top row are one-component designs (PDB 4EGG & 3VCD, left to right) [16], with each trimer shown in a different color for clarity. The middle and bottom rows (PDB 4NWR, 4NWP, 4NWO, 4NWN, & 4ZK7, left to right and top to bottom) are two-component designs [23••,24]. Natural dimers are in yellow. Natural trimers are in blue and orange to differentiate the different trimers in the same design. Right panel: crystal structures of 2-component icosahedra obtained by all three possible symmetry combinations (PDB 5IM5 (I:5 + 3), 5IM4 (I:5 + 2), and 5IM6 (I:3 + 2)) [25••]. Letters on the top left corner of the structure indicates the symmetry type (T: tetrahedral, O: octahedral, I: icosahedral). The numbers in the annotation indicate the component symmetry types. Where present, parenthetical values indicate the symmetry of the main designed interface.

been demonstrated in recent work [56••,57•], but a complete set of geometric rules for generating two and three-dimensional materials has not been articulated previously. In order to promote further studies, we provide a list of the allowable symmetry combinations here (Table 1).

Two-dimensional layers can be of two different classes depending on their sidedness, or lack thereof. When two symmetry axes that are both perpendicular to the layer are combined, and at least one of those axes is of higher order than a 2-fold, the result is a layer with distinguishable sides (i.e. a distinct top and bottom). In that sense such layers are oriented. The allowable symmetry combinations for oriented layers are 2 + 3, 2 + 4, 2 + 6, 3 + 3, 3 + 6, and 4 + 4 (Table 1). Layered structures of the other class, where the top and bottom of the layer are indistinguishable, arise whenever one or both of the two symmetries being combined carries a 2-fold axis of symmetry in the plane of the layer. A total of 33 layer designs are possible (Table 1).

Much of the design space for designed layers is unexplored, but a few recent successes have been reported. Small two-dimensional assemblies with limited range order were described in early work by Ringler *et al.* by doubly biotinylating the subunits of aldolase (a C4 tetramer) and then assembling those tetramers using streptavidin (a D2 tetramer with a biotin binding site in each

**Table 1**

**Multiplication table for designing self-assembling protein materials from combinations of two simpler symmetric components or interfaces.**[a]

| x | C2 | C3 | C4 | C6 | D2 | D3 | D4 | D6 | T | O |
|---|---|---|---|---|---|---|---|---|---|---|
| **C2** | b | D3, T, O, I p6, p321 $I2_13$, $P4_132$ | D4, O p4, $p42_12$ I432 | D6 p6, p622 | c222, p422, p622 $I4_122$, $P6_222$, I432, $I4_132$ | p312, p622 R32, $P6_322$, $F4_132$, $I4_132$, I432, $P4_132$ | p422 I422, P432, I432 | p622 P622 | P23, F23, $F4_132$ | P432, F432, I432 |
| **C3** | ■ | T p3 $P2_13$ | O F432 | p6 | p622 P23, F432, $I4_132$ | p321, p312 $P4_132$ | P432 | p622 | F23 | F432 |
| **C4** | ■ | ■ | p4 P432 | ▨ | p422, $p42_12$ I432, F432 | I432 | p422 P432 | ▨ | F432 | P432 |
| **C6** | ■ | ■ | ■ |  | p622 | p622 | ▨ |  | ▨ | ▨ |
| **D2** | ■ | ■ | ■ | ■ | p222, p622 F222, $P4_222$, $P6_222$, $P4_232$, $I4_132$ | p622 P622, $P4_232$, $I4_132$ | p422 P422, I422, I432 | p622 P622 | P23, F432, $P4_232$ | F432, I432 |
| **D3** | ■ | ■ | ■ | ■ | ■ | p321 P312, $P6_322$, $P4_232$, $P4_132$ | I432 | p622 P622 | $F4_132$ | I432 |
| **D4** | ■ | ■ | ■ | ■ | ■ | ■ | p422 P422, P432 | ▨ |  | P432 |
| **D6** | ■ | ■ | ■ | ■ | ■ | ■ | ■ |  | ▨ | ▨ |
| **T** | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | F23 | F432 |
| **O** | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | P432, F432 |

Finite assemblies (point group symmetries) are indicated in the blue font. 2-D layers are indicated in red, 3-D crystalline arrays in purple. In many cases, two component symmetries can be combined in different geometries giving rise to distinct symmetry types. Gray boxes indicate symmetry combinations that are disallowed mathematically. A few symmetry combinations are not formally disallowed but are not amenable to design using compact building blocks. Whenever a chiral space group appears (*e.g.* $P4_132$), its enantiomer (*e.g.* $P4_321$) is also possible but is not listed here for brevity.

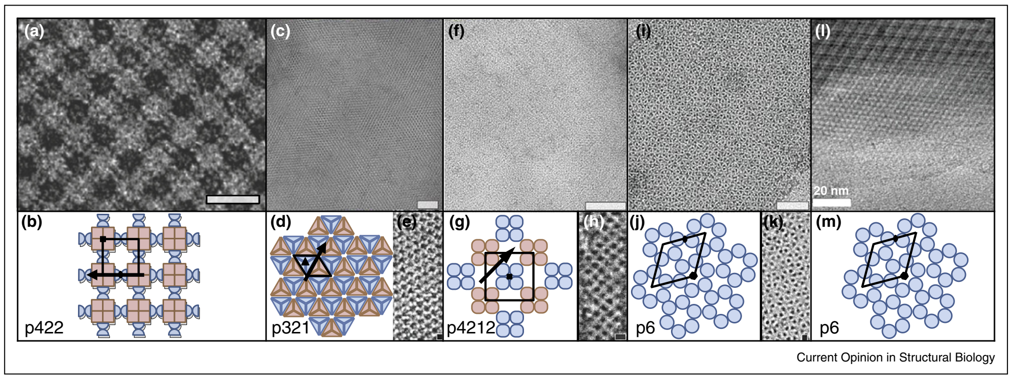
[a] Additional possibilities exist but are not listed here for arrangements where more than two component symmetries are combined, or where one of the component symmetries is a screw axis of rotation.

‡ Non-intersecting 2-fold axes give rise to linear or helical filaments. Linear assemblies, including rod groups, are not included here.

subunit) [27]. If long range order had been achieved in this case, the result would have corresponded to layer symmetry p422. Longer range order in designed layers was demonstrated by Sinclair *et al.* [22]. There the best case was obtained by combining the D2 tetrameric streptavidin with a D4 octameric protein that had been biotinylated. The relative orientations of the components in that case were not specifically designed and could have produced other results, but a two-dimensional layer with p422 symmetry was obtained (Figure 3). More recently, computational interface design was used to create specifically defined protein layers of a few different types [56••]. A 2-fold interface was designed between natural C6 hexameric units to give layer symmetry p6, between natural C4 tetrameric units to give layer symmetry $p42_12$, and between natural C3 trimeric units to give layer symmetry p321 (Figure 3). In another variation, two copies of a C6 hexameric protein were fused in tandem in such a way that the short linker between them, along with a designed 2-fold interface, led to a pseudo-p6 layer [57•]. In a most recent study, C4 symmetric units were connected through 2-fold interactions on the basis of engineered cysteine disulfide bonds or engineered metal binding sites [37•]. Multiple kinds of symmetric outcomes are possible from such a combination, and 2-D layers belonging to p4 and $p42_12$ symmetry types were obtained.

For designing extended three-dimensional crystalline materials, the possibilities are even more expansive, and the design rules are not so obvious. Defining exactly what symmetry type would be generated from a combination of two separate symmetries relies on two considerations. The overall rotational symmetry of the resulting material is given by the (group) product of the rotational symmetries of the two components; this is a relatively straightforward issue. A somewhat more complex problem is discerning the correct outcome among a set of candidate space groups once the underlying rotational

**Figure 3**



Electron micrographs of protein layers designed to assemble with high symmetry and showing long-range order. Symmetry diagrams are shown under each micrograph, accompanied in some cases by enlarged images. Each symmetry diagram shows the repeating unit cell within which one instance of each of the component symmetry elements is indicated using standard symbols: black arrows, 2-fold symmetry axes in the plane of the layer; black ovals, black triangles, black squares, and black hexagons indicate 2-fold, 3-fold, 4-fold, and 6-fold axes perpendicular to the plane of the layer, respectively. **(a)** a p422 layer formed by combining D4 and D2 symmetry components; **(c)** a p321 layer formed by combining C3 trimers with a 2-fold de novo interface; **(f)** a p42$_1$2 layer formed by combining C4 tetramers with a 2-fold de novo interface; (i) a p6 layer formed by combining C6 hexamers and a 2-fold de novo interface; **(l)** a pseudo p6 layer formed by combining C6 hexamers using a covalent fusion and a pseudo 2-fold de novo interface. (e, h, k) enlarged images of (c), (f), (i), respectively. Scale bars: (a) — 20 nm; (c) — 50 nm; (f) — 50 nm; (i) — 50 nm; (l) — 20 nm.
Images reproduced with permission from: panel (a) — Sinclair *et al.* [22]; panels (c, e, f, h, i, and k) — Gonen *et al.* [56••]; panel (l) — Matthaei *et al.* [57•]

symmetry is known. The correct choice can generally be ascertained from the standard tables of crystallographic space groups by identifying the one having Wyckoff positions with symmetries corresponding to those of the two components being combined. In total, more than 70 distinct types of 3D materials are possible within the scheme of combining two types of rotation (point group) symmetries. The majority of them have underlying cubic (T or O) symmetries and are therefore isotropic, while the others are dihedral.

On the experimental side, the space of designed protein crystals is mostly unexplored, but there are a few early examples. As discussed above, Lanci *et al.* [19] designed a coiled-coil peptide to form P6 space group symmetry, and Sontz *et al.* [36••] combined a ferritin, into which a metal side had been engineered, with a bivalent bridging compound to form a crystal whose space group symmetry was pseudo-I432. Sinclair *et al.* used their fusion method to form three-dimensional solid materials, but without sufficient order to confirm the intended crystalline packing by X-ray diffraction [22]. Beyond these few examples, the area of designed protein crystals remains open. The possible applications for such materials are diverse: creating materials with a very high density of reactive/catalytic groups or recognition motifs, and conferring specific physical properties on target proteins, including spacing, dimensionality, porosity, and solid-phase separability from solution components.

## Variations, challenges and future directions

The ideas and rules formulated here are somewhat narrowly constructed. They represent the simplest design routes (i.e. the minimum requirements) for construction by symmetric assembly. Broader outcomes are possible if interfaces between components are designed in different ways. For example, King's single-component designed interface method [18] allows for symmetries not accounted for in Table 1 if a new contact between like oligomers creates a screw axis of symmetry instead of a pure rotation. Ultimately, in the absence of limits on designing protein–protein interfaces, any symmetric architecture could be designed, including those that require larger numbers of distinct interaction types. The three-dimensional crystal in space group P6 designed by Lanci *et al.* [19] is a case in point using relatively simple building blocks. In addition, other approaches for designing large protein and peptide-based structures have been developed that rely less strictly, or in different ways, on symmetry. Fletcher *et al.* [58•] combined a homotrimeric coiled-coil and a heterodimeric coiled-coil that interact with each other. This resulted in unilamellar spheres approximately 100 nanometers in diameter, with overall structures that were not exactly symmetric though assembly was driven by local symmetry.

Doll *et al.* [59] also combined coiled-coil sequences with different symmetric properties (5-fold and 3-fold) to produce roughly spherical clusters having sizes consistent with icosahedral assembly. In a distinctly different line of attack, Gradišar *et al.* demonstrated the construction of a tetrahedral architecture on the basis of asymmetric interactions between coiled coil motifs in a long, designed protein molecule whose folding pattern traverses the edges of the entire polyhedron twice [60].

The different design strategies discussed here present their own advantages and challenges. The initial design strategy of helical fusions between oligomers presents a relatively low barrier in the sense of not needing to create de novo interfaces, but flexibility creates an obvious challenge. This sometimes leads to alternate assembly outcomes [17•,61]. Strategies involving more rigid linkers could improve the reliability of this method. A recent report demonstrated some success in rigidifying a continuous alpha helical linker between two protein components by a specific chemical cross-link between cysteines at positions $i$ and $i + 11$ [62]. Another approach to rigidification would be to use a coiled-coil linker as the motif joining two separate oligomeric units, with a single helix of a hetero coiled-coil motif extending from each of the oligomeric components. These are avenues of ongoing study.

The main challenges with methods based on de novo interface design relate to misfolding or unintended assembly — often insoluble aggregation — most likely caused by introduction of new regions of hydrophobicity in a protein surface. The success rates for geometrically specific interface design in the context of symmetric assemblies is currently in the range of about 10%. One case has been reported where a failed design could be rescued by increasing the charge on the protein molecule [24]. This suggests that optimizing certain design parameters and selection criteria might substantially increase the success rates. Another notable trend is the increasing success in designing novel protein folds and assemblies by focusing on building blocks that are all (or mainly) alpha helical or otherwise repetitive in structure [63–68]. New strategies for high throughput selection or screening of designs for correct assembly could also be impactful.

As design strategies continue to improve, and with construction rules in hand for building wide-ranging types of symmetric architectures, the coming years should bring a rich diversity of new protein based materials with useful applications.

## Conflict of interest

The authors declare no competing interests.

## Acknowledgements

This work was supported by NSF grant CHE-1332907. The authors thank Dan McNamara, Yen-Ting Lai, Kevin Cannon and other members of the Yeates lab for their ideas, and members of the David Baker laboratory, including Jacob Bale and Neil King, for access to their designs ahead of publication. We thank Frank DiMaio and Will Sheffler for discussions on symmetry rules and Dek Woolfson for discussions related to coiled-coil polypeptides.

## References and recommended reading

Papers of particular interest, published within the period of review, have been highlighted as:

- • of special interest
- •• of outstanding interest

1. Goodsell DS, Olson AJ: **Structural symmetry and protein function**. *Annu Rev Biophys Biomol Struct* 2000, **29**:105-153.
2. Marsh JA, Teichmann SA: **Structure, dynamics assembly, and evolution of protein complexes**. *Annu Rev Biochem* 2015, **84**:551-575.
3. Yeates TO, Thompson MC, Bobik TA: **The protein shells of bacterial microcompartment organelles**. *Curr Opin Struct Biol* 2011, **21**:223-231.
4. Seeman NC: **Nucleic acid junctions and lattices**. *J Theor Biol* 1982, **99**:237-247.
5. Jones MR, Seeman NC, Mirkin CA: **Nanomaterials. Programmable materials and the nature of the DNA bond**. *Science (New York, N.Y.)* 2015, **347**:1260901.
6. Ispolatov I, Yuryev A, Mazo I, Maslov S: **Binding properties and evolution of homodimers in protein–protein interaction networks**. *Nucl Acids Res* 2005, **33**:3629-3635.
7. Pereira-Leal JB, Levy ED, Kamp C, Teichmann SA: **Evolution of protein complexes by duplication of homomeric interactions**. *Genom Biol* 2007, **8**:R51.
8. Crick FH, Watson JD: **Structure of small viruses**. *Nature* 1956, **177**:473-475.
9. Pandya MJ, Spooner GM, Sunde M, Thorpe JR, Rodger A, Woolfson DN: **Sticky-end assembly of a designed peptide fiber provides insight into protein fibrillogenesis**. *Biochemistry* 2000, **39**:8728-8734.
10. Ogihara NL, Ghirlanda G, Bryson JW, Gingery M, DeGrado WF, Eisenberg D: **Design of three-dimensional domain-swapped dimers and fibrous oligomers**. *Proc Natl Acad Sci USA* 2001, **98**:1404-1409.
11. Robertson DE, Farid RS, Moser CC, Urbauer JL, Mulholland SE, Pidikiti R, Lear JD, Wand AJ, DeGrado WF, Dutton PL: **Design and synthesis of multi-haem proteins**. *Nature* 1994, **368**:425-432.
12. Kuhlman B, O'Neill JW, Kim DE, Zhang KY, Baker D: **Conversion of monomeric protein L to an obligate dimer by computational protein design**. *Proc Natl Acad Sci USA* 2001, **98**:10687-10691.
13. Wukovitz SW, Yeates TO: **Why protein crystals favour some space-groups over others**. *Nat Struct Biol* 1995, **2**:1062-1067.
14. Padilla JE, Colovos C, Yeates TO: **Nanohedra: Using symmetry to design self assembling protein cages, layers, crystals, and filaments**. *Proc Natl Acad Sci USA* 2001, **98**:2217-2221.
15. Lai Y-T, Cascio D, Yeates TO: **Structure of a 16-nm Cage Designed by Using Protein Oligomers**. *Science* 2012, **336**:1129.
16. Lai Y-T, Tsai K-L, Sawaya MR, Asturias FJ, Yeates TO: **Structure and flexibility of nanoscale protein cages designed by symmetric self-assembly**. *J Am Chem Soc* 2013, **135**:7738-7743.
17. • Lai Y-T, Reading E, Hura GL, Tsai K-L, Laganowsky A, Asturias FJ, Tainer JA, Robinson CV, Yeates TO: **Structure of a designed protein cage that self-assembles into a highly porous cube**. *Nat Chem* 2014, **6**:1065-1071.

Using the alpha helical oligomer fusion strategy, the authors reported the design and crystallographic validation of a highly porous 24-subunit cubic assembly with a large central cavity, demonstrating the feasibility of controlling geometry over long distance while providing porosity. Helix flexibility admitted alternative assembly forms.

18. King NP, Sheffler W, Sawaya MR, Vollmar BS, Sumida JP, André I, Gonen T, Yeates TO, Baker D: **Computational design of self-assembling protein nanomaterials with atomic level accuracy**. *Science* 2012, **336**:1171-1174.

19. Lanci CJ, MacDermaid CM, Kang S-g, Acharya R, North B, Yang X, Qiu XJ, DeGrado WF, Saven JG: **Computational design of a protein crystal**. *Proc Natl Acad Sci USA* 2012, **109**:7304-7309.

20. Grueninger D, Treiber N, Ziegler MOP, Koetter JWA, Schulze M-S, Schulz GE: **Designed protein–protein association**. *Science (New York, N.Y.)* 2008, **319**:206-209.

21. DiMaio F, Leaver-Fay A, Bradley P, Baker D, André I: **Modeling symmetric macromolecular structures in rosetta3**. *PLoS One* 2011, **6**:e20450.

22. Sinclair JC, Davies KM, Vénien-Bryan C, Noble MEM: **Generation of protein lattices by fusing proteins with matching rotational symmetry**. *Nat Nanotechnol* 2011, **6**:558-562.

23. King NP, Bale JB, Sheffler W, McNamara DE, Gonen S, Gonen T,
•• Yeates TO, Baker D: **Accurate design of co-assembling multi-component protein nanomaterials**. *Nature* 2014, **510**:103-108.
Using improved algorithms in the Rosetta software suite, the authors illustrate the design and validation of a series of novel protein cages wherein each cage assembles from two distinct oligomeric components. A de novo interface between the two types of oligomers drives the symmetric assembly.

24. Bale JB, Park RU, Liu Y, Gonen S, Gonen T, Cascio D, King NP, Yeates TO, Baker D: **Structure of a designed tetrahedral protein assembly variant engineered to have improved soluble expression**. *Protein Sci* 2015, **24**:1695-1701.

25. Bale JB, Gonen S, Liu Y, Sheffler W, Ellis D, Thomas C, Cascio D,
•• Yeates TO, Gonen T, King NP *et al.*: **Accurate design of megadalton-scale two-component icosahedral protein complexes**. *Science* 2016, **353**:389-394.
This study demonstrates the design and atomic level validation of the largest geometrically specific protein cages to date using a two-component strategy and interface design. Controlled encapsulation of cargo proteins by charge complimentarity is demonstrated.

26. Dotan N, Arad D, Frolow F, Freeman A: **Self-assembly of a tetrahedral lectin into predesigned diamondlike protein crystals**. *Angew Chem (Int Ed. in English)* 1999, **38**:2363-2366.

27. Ringler P, Schulz GE: **Self-assembly of proteins into designed networks**. *Science* 2003, **302**:106-109.

28. Salgado EN, Ambroggio XI, Brodin JD, Lewis RA, Kuhlman B, Tezcan FA: **Metal templated design of protein interfaces**. *Proc Natl Acad Sci USA* 2010, **107**:1827-1832.

29. Salgado EN, Radford RJ, Tezcan FA: **Metal-directed protein self-assembly**. *Acc Chem Res* 2010, **43**:661-672.

30. Salgado EN, Lewis RA, Mossin S, Rheingold AL, Tezcan FA: **Control of protein oligomerization symmetry by metal coordination – C2 and C3 symmetrical assemblies through Cu(II) and Ni(II) coordination**. *Inorg Chem* 2009, **48**:2726-2728.

31. Brodin JD, Ambroggio XI, Tang C, Parent KN, Baker TS, Tezcan FA: **Metal-directed, chemically tunable assembly of one-, two- and three-dimensional crystalline protein arrays**. *Nat Chem* 2012, **4**:375-382.

32. Salgado EN, Lewis RA, Faraone-Mennella J, Tezcan FA: **Metal-mediated self-assembly of protein superstructures: influence of secondary interactions on protein oligomerization and aggregation**. *J Am Chem Soc* 2008, **130**:6082-6084.

33. Salgado EN, Faraone-Mennella J, Tezcan FA: **Controlling protein−protein interactions through metal coordination: assembly of a 16-helix bundle protein**. *J Am Chem Soc* 2007, **129**:13374-13375.

34. Laganowsky A, Zhao M, Soriaga AB, Sawaya MR, Cascio D, Yeates TO: **An approach to crystallizing proteins by metal-mediated synthetic symmetrization**. *Protein Sci* 2011, **20**:1876-1890.

35. Leibly DJ, Arbing MA, Pashkov I, DeVore N, Waldo GS, Terwilliger TC, Yeates TO: **A suite of engineered GFP molecules for oligomeric scaffolding**. *Structure* 2015, **23**:1754-1768.

36. Sontz PA, Bailey JB, Ahn S, Tezcan FA: **A metal organic framework**
•• **with spherical protein nodes: rational chemical design of 3D protein crystals**. *J Am Chem Soc* 2015, **137**:11598-11601.
The authors constructed a three-dimensional protein crystal with a prescribed lattice by introducing metal–organic linker interactions between adjacent ferritin cages, which are naturally cubic/octahedral. This is the first report of a designed metal–organic mediated 3-D protein crystal. Its designed structure was shown to be accurate by X-ray diffraction at atomic resolution.

37. Suzuki Y, Cardone G, Restrepo D, Zavattieri PD, Baker TS,
• Tezcan FA: **Self-assembly of coherently dynamic, auxetic, two-dimensional protein crystals**. *Nature* 2016, **533**:369-373.
The authors obtain multiple well-ordered two-dimensional arrays from a cyclic tetrameric protein engineered to make 2-fold contacts with itself via disulfide or metal binding motifs. The layers exhibit shrinking/expansion transitions.

38. Sakai F, Yang G, Weiss MS, Liu Y, Chen G, Jiang M: **Protein crystalline frameworks with controllable interpenetration directed by dual supramolecular interactions**. *Nat Commun* 2014, **5**:4634.

39. Mou Y, Yu J-Y, Wannier TM, Guo C-L, Mayo SL: **Computational design of co-assembling protein-DNA nanowires**. *Nature* 2015, **525**:230-233.

40. Brodin JD, Auyeung E, Mirkin CA: **DNA-mediated engineering of**
• **multicomponent enzyme crystals**. *Proc Natl Acad Sci U S A* 2015, **112**:4564-4569.
The authors create hybrid protein-DNA crystalline materials by chemically coating proteins with DNA and then adding bridging DNA molecules to drive association between two different types of protein particles. DNA-coated gold nanoparticles were also used.

41. Lai Y-T, King NP, Yeates TO: **Principles for designing ordered protein assemblies**. *Trends Cell Biol* 2012, **22**:653-661.

42. Sciore A, Su M, Koldewey P, Eschweiler JD, Diffley KA,
• Linhares BM, Ruotolo BT, Bardwell JCA, Skiniotis G, Marsh EN: **A flexible, symmetry-directed approach to assembling protein cages**. *Proc Natl Acad Sci USA* 2016. (in press).
The authors create a 24-subunit assembly by fusing a trimeric protein to a tetrameric coiled-coil motif.

43. Hsia Y, Bale JB, Gonen S, Shi D, Sheffler W, Fong KK,
• Nattermann U, Xu C, Huang PS, Ravichandran R *et al.*: **Design of a hyperstable 60-subunit protein icosahedron**. *Nature* 2016, **535**:136-139.
The authors create a 60-subunit icosahedral cage using interface design and a one-component strategy.

44. Yeates TO, Padilla JE: **Designing supramolecular protein assemblies**. *Curr Opin Struct Biol* 2002, **12**:464-470.

45. Doll TAPF, Raman S, Dey R, Burkhard P: **Nanoscale assemblies and their biomedical applications**. *J R Soc Interface* 2013:10.

46. López-Sagaseta J, Malito E, Rappuoli R, Bottomley MJ: **Self-assembling protein nanoparticles in the design of vaccines**. *Comput Struct Biotechnol J* 2015, **14**:58-68.

47. Howorka S: **Rationally engineering natural protein assemblies in nanobiotechnology**. *Curr Opin Biotechnol* 2011, **22**:485-491.

48. Wörsdörfer B, Woycechowsky KJ, Hilvert D: **Directed evolution of a protein container**. *Science (New York, N.Y.)* 2011, **331**:589-592.

49. Patterson DP, Schwarz B, El-Boubbou K, Oost Jvd, Prevelige PE, Douglas T: **Virus-like particle nanoreactors: programmed encapsulation of the thermostable CelB glycosidase inside the P22 capsid**. *Soft Matter* 2012, **8**:10158-10166.

50. Champion CI, Kickhoefer VA, Liu G, Moniz RJ, Freed AS, Bergmann LL, Vaccari D, Raval-Fernandes S, Chan AM, Rome LH *et al.*: **A vault nanoparticle vaccine induces protective mucosal immunity**. *PLOS One* 2009, **4**:e5409.

51. Han M, Kickhoefer VA, Nemerow GR, Rome LH: **Targeted vault nanoparticles engineered with an endosomolytic peptide deliver biomolecules to the cytoplasm**. *ACS Nano* 2011, **5**:6128-6137.

52. Kar UK, Jiang J, Champion CI, Salehi S, Srivastava M, Sharma S, Rabizadeh S, Niazi K, Kickhoefer V, Rome LH *et al.*: **Vault nanocapsules as adjuvants favor cell-mediated over antibody-mediated immune responses following immunization of mice**. *PLoS One* 2012, **7**:e38553.

53. Zschoche R, Hilvert D: **Diffusion-Limited Cargo Loading of an Engineered Protein Container**. *J Am Chem Soc* 2015, **137**:16121-16132.

54. Correia BE, Bates JT, Loomis RJ, Baneyx G, Carrico C, Jardine JG, Rupert P, Correnti C, Kalyuzhniy O, Vittal V *et al.*: **Proof of principle for epitope-focused vaccine design**. *Nature* 2014, **507**:201-206.

55. Sliepen K, Ozorowski G, Burger JA, van Montfort T, Stunnenberg M, LaBranche C, Montefiori DC, Moore JP, Ward AB, Sanders RW: **Presenting native-like HIV-1 envelope trimers on ferritin nanoparticles improves their immunogenicity**. *Retrovirology* 2015, **12**:82.

56. •• Gonen S, DiMaio F, Gonen T, Baker D: **Design of ordered two-dimensional arrays mediated by noncovalent protein–protein interfaces**. *Science* 2015, **348**:1365-1368.
The authors report on three examples of symmetric protein layers designed by introducing a de novo interface between symmetry-related copies of a natural cyclic oligomer. The layers showed long range order.

57. • Matthaei JF, DiMaio F, Richards JJ, Pozzo LD, Baker D, Baneyx F: **Designing two-dimensional protein arrays through fusion of multimers and interface mutations**. *Nano Lett* 2015 http://dx.doi.org/10.1021/acs.nanolett.5b01499.
The authors create an ordered 2-D array by genetically fusing two monomers of a cyclic hexameric protein in an orientation that drives extended rather than closed assembly

58. • Fletcher JM, Harniman RL, Barnes FRH, Boyle AL, Collins A, Mantell J, Sharp TH, Antognozzi M, Booth PJ, Linden N *et al.*: **Self-assembling cages from coiled-coil peptide modules**. *Science* 2013, **340**:595-599.
The authors show that a self-assembling material can be designed by linking trimeric and dimeric coiled-coils through disulfide bonds. Hexagonal networks form, which then close up to form roughly spherical nanoparticles with diameters in the 100 nm range.

59. Doll TAPF, Dey R, Burkhard P: **Design and optimization of peptide nanoparticles**. *J Nanobiotechnol* 2015, **13**:73.

60. Gradišar H, Božič S, Doles T, Vengust D, Hafner-Bratkovič I, Mertelj A, Webb B, Šali A, Klavžar S, Jerala R: **Design of a single-chain polypeptide tetrahedron assembled from coiled-coil segments**. *Nat Chem Biol* 2013, **9**:362-366.

61. Lai Y-T, Jiang L, Chen W, Yeates TO: **On the predictability of the orientation of protein domains joined by a spanning alpha-helical linker**. *Protein Eng Des Select* 2015, **28**:491-499.

62. Jeong WH, Lee H, Song DH, Eom J-H, Kim SC, Lee H-S, Lee H, Lee J-O: **Connecting two proteins using a fusion alpha helix stabilized by a chemical cross linker**. *Nat Commun* 2016, **7**:11031.

63. Boyken SE, Chen Z, Groves B, Langan RA, Oberdorfer G, Ford A, Gilmore JM, Xu C, DiMaio F, Pereira JH *et al.*: **De novo design of protein homo-oligomers with modular hydrogen-bond network-mediated specificity**. *Science* 2016, **352**:680-687.

64. Brunette TJ, Parmeggiani F, Huang PS, Bhabha G, Ekiert DC, Tsutakawa SE, Hura GL, Tainer JA, Baker D: **Exploring the repeat protein universe through computational protein design**. *Nature* 2015, **528**:580-584.

65. Huang PS, Oberdorfer G, Xu C, Pei XY, Nannenga BL, Rogers JM, DiMaio F, Gonen T, Luisi B, Baker D: **High thermodynamic stability of parametrically designed helical bundles**. *Science* 2014, **346**:481-485.

66. Thomson AR, Wood CW, Burton AJ, Bartlett GJ, Sessions RB, Brady RL, Woolfson DN: **Computational design of water-soluble alpha-helical barrels**. *Science* 2014, **346**:485-488.

67. Jacobs TM, Williams B, Williams T, Xu X, Eletsky A, Federizon JF, Szyperski T, Kuhlman B: **Design of structurally distinct proteins using strategies inspired by evolution**. *Science* 2016, **352**:687-690.

68. Glover DJ, Giger L, Kim SS, Naik RR, Clark DS: **Geometrical assembly of ultrastable protein templates for nanomaterials**. *Nat Commun* 2016, **7**:11771.

# CHAPTER 2

# Geometric lessons and design strategies for nanoscale protein cages

# Geometric Lessons and Design Strategies for Nanoscale Protein Cages

Joshua Laniado[1]*, Kevin A. Cannon[1]*, Justin E. Miller[2], Michael R. Sawaya[3], Dan E. McNamara[2] and Todd O. Yeates[1,2,3]

[1]Molecular Biology Institute, UCLA
[2]UCLA-DOE Institute for Genomics and Proteomics
[3]UCLA Department of Chemistry and Biochemistry

*equal author contributions

Keywords: Protein Design, Nanotechnology, Symmetry, Self-Assembly, Coiled-Coil

## ABSTRACT

Protein molecules bring rich functionality to the field of designed nanoscale architectures. High symmetry protein cages are rapidly finding diverse application in biomedicine, nanotechnology and imaging, but methods for their reliable and predictable construction remain challenging. In this study we introduce an approach for designing protein assemblies that combines new ideas with favorable elements adapted from recent work. Cubically symmetric cages can be created by combining two simpler symmetries, following recently established principles. Here, two different oligomeric protein components are brought together in a geometrically specific arrangement by their separate genetic fusion to individual components of a heterodimeric coiled-coil polypeptide motif of known structure. Fusions between components are made by continuous alpha helices to limit flexibility. After computational design, we tested ten different protein cage constructions experimentally, two of which formed larger assemblies. One produced the intended octahedral cage, approximately 26 nm in diameter, while the other appeared to produce the intended tetrahedral cage as a minor component, crystallizing instead in an alternate form representing a collapsed structure of lower stoichiometry and symmetry. Geometric distinctions between the two characterized designs help explain the different degrees of success, leading to clearer principles and improved prospects for the routine creation of novel nanoscale protein architectures using diverse methods.

## INTRODUCTION

In the last decade, developments in the field of protein design have produced a growing suite of novel protein assemblies unseen in nature[1–14]. Highly symmetric, cubic and icosahedral protein cages have drawn particular attention. Early studies laid out the essential design requirements[15]. When two (or more) symmetric objects (e.g. simple protein oligomers) are held together in a geometrically specific way, a larger and more complex architecture can be produced by self-assembly. The resulting symmetry is dictated by the combined symmetries of the two underlying components. For the design of high symmetry protein cages, the symmetry axes belonging to the component oligomers must intersect at an angle according to the Platonic solids (i.e. tetrahedron, cube, icosahedron). An example would be a homotrimer and a homodimer that are positioned by design such that their individual symmetry axes intersect at an angle of 54.7° at the center of a tetrahedron. While the geometric requirements for assembling diverse protein materials using the principle of symmetry combinations have been studied for nearly 20 years[2,7,8,15–19], the practical matter of how to engineer a strictly defined geometric relationship between two complex protein molecules remains challenging, relying on advanced molecular engineering and computational design methods.

A successful method for holding two protein oligomers in a rigid orientation with respect to one another was demonstrated in 2001, when Padilla and coworkers used an α-helical linker to bring together a dimer and a trimer in order to form a protein tetrahedron[15,20]. This design strategy utilizes an α-helical linker fused between terminal helices of the component oligomers, creating a continuous intervening helix rigid enough to hold the oligomers in the correct orientation[21]. Reliable application of this strategy tends to be challenged by helix flexibility[22], but the approach has successfully produced a protein cube[23] and icosahedron[24] in addition to the original tetrahedron.

Other strategies for orienting oligomers to self-assemble into cages have been demonstrated as well. More than a dozen protein cages have been characterized at near-atomic resolution by X-ray crystallography[17,20,23,25,26], with several others validated by other techniques such as electron microscopy (EM)[3,10–13,27,28]. Most of the designs that have led to crystal structures were the result of computational interface design *via* the Rosetta protein design software[17,25,26]. In this method, natural protein oligomers are first computationally docked such that their symmetry axes have the desired angle of intersection for a specific point group symmetry (T, O, or I). Then mutations that could lead to a new interface between oligomers in the specified orientation are suggested computationally by searching for amino acids substitutions that give favorable calculated binding energies. Numerous cages have been successfully designed by the interface design approach developed by King *et al.*[17], yet such methods are highly demanding computationally, and experimental success rates tend to be limited by imperfect knowledge of how to produce tight binding interfaces (which are often relatively hydrophobic) without causing non-specific assembly – either aberrant structures or aggregation into inclusion bodies – when overexpressed in bacteria. Recent efforts to lower the hydrophobicity and improve hydrogen bonding in designed interfaces are improving the performance of interface design approaches[29–31]. Other studies have retained the idea of using genetic fusion to connect oligomeric components but have sought to relax the requirement for rigidity imposed by a continuous helical connection. Flexible connections between oligomers have been used to form ordered 2-D protein layers[19] and several types of protein clusters or cages[32], some with approximate T, O, and I symmetry[10–12]. Methods based on flexible connections have produced assemblies of the intended forms, but they have largely evaded detailed structural characterization by crystallography or high resolution cryo-EM. Protein cages designed by various methods are beginning to find applications in areas as diverse as enzyme scaffolding,[33,34] vaccine design,[35] nucleic acid encapsulation,[36,37] and imaging[38,39]. Thus, further exploration of design methodologies could facilitate more routine production of novel protein cages with applications in biomedicine and nanotechnology.

In the present study, we built on the original concept of joining symmetric proteins based on continuous α helices, but with a variation intended to reduce the innate flexibility of a single α helix. This new approach effectively replaces the single α-helical linker of the original fusion design method with a (presumptively) more rigid heterodimeric α helical coiled-coil; one component of the coiled-coil is fused to either oligomeric subunit. Like previous fusion methods, the computational design demands are modest. Similar to methods based on interface design, the resulting architectures are held together by a non-covalent association between proteins, but here that association (i.e. between coiled coil components) is known in advance. We present the structural characterization of two different cages designed by this method, with some unexpected results that demonstrate crucial lessons about protein cage design principles.

## RESULTS

### Protein design methodology

In this work we set out to design 3-dimensional protein cages obeying tetrahedral (T) or octahedral (O) point group symmetry, which would self-assemble from two different oligomeric protein components *via* fusion to heterodimeric coiled coils. The general design workflow closely follows that of previous helical fusion cage designs, with some variations (see Methods). The ultimate shape of the self-assembled cage construct is determined by the rotational symmetries of the component oligomers and the angle of intersection between their symmetry axes. Combinations of trimeric and dimeric components with their symmetry axes intersecting at an angle of 54.7°, for example, should self-assemble with 12 copies of each subunit ($a_{12}b_{12}$) to form a tetrahedron (Fig 1, top panel). On the other hand, combinations of tetrameric and trimeric components intersecting at 54.7° should self-assemble with 24 copies of each subunit ($a_{24}b_{24}$) to build an octahedron (Fig 1, bottom panel). Designs bearing T or O symmetry were generated computationally. All possible pairs of candidate dimers and trimers or tetramers and trimers in the PDB protein structure database were considered. For every candidate pair, a subunit from each oligomer was computationally fused through its alpha helical terminus to one component of the c-Fos/c-Jun heterodimeric coiled-coil structure. Following that step, candidate constructions where the symmetry axes carried by the two component oligomers intersected at or nearly at the required angle of 54.7° were identified. The favorable geometric candidates were then screened for steric clashes that might occur upon symmetry expansion. A final visual inspection was carried out and potentially problematic cases (e.g. membrane proteins) were removed. Ten designs were chosen for experimental characterization.

The design approach used here takes advantage of non-covalent interfacial interactions (between the helices of the hetero-dimeric coiled coil) to hold the two symmetric oligomeric protein components together, as opposed to the genetic fusion or computational interface design strategies used for other cages that have been reported. Coiled-coil motifs have been widely used for protein design, with considerable success and well-characterized principles[4,5,10–12,40–45]. For our coiled-coil motif we used an engineered version of the c-Fos/c-Jun coiled coil (see Supplement)[41,46,47]. Our coiled-coil interaction serves a similar role as the computationally designed interfaces in King's 2014 work[25], namely to bring two different oligomeric components together in a precisely defined fashion. The distinction is that here, instead of requiring novel interface design for each computational candidate, the non-covalent interaction provided by c-Jun/c-Fos is understood in advance.  As with King, the two-component approach allows for hierarchical assembly (e.g. of complete oligomers) *en route* to formation of a complete cage, with potential benefits for robust assembly, compared to approaches where a single polypeptide chain embodies two different oligomerizing domains.

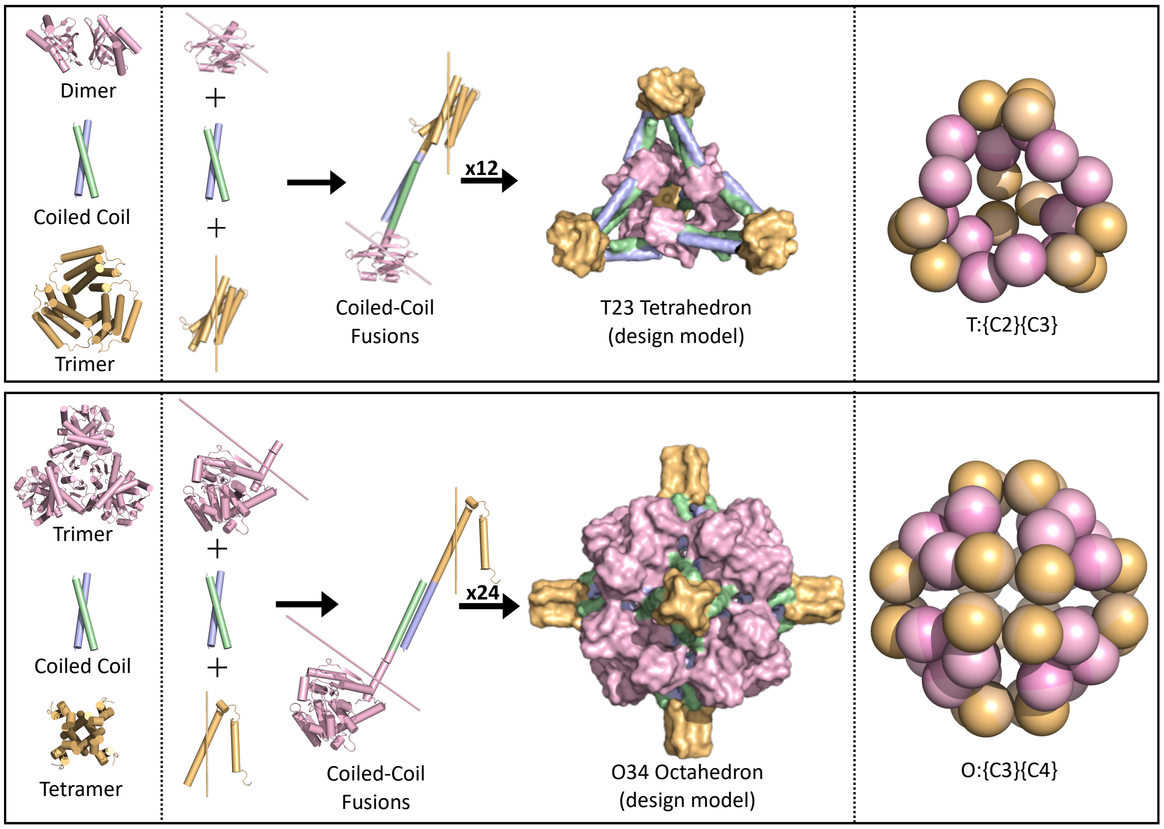


**Figure 1.** Design of two-component protein cages using a coiled-coil helical fusion-based approach. Two distinct oligomeric subunits are fused to separate helices of a heterodimeric coiled-coil linker (blue and green). The T23 tetrahedral design (top panel) was constructed by combining a C2 dimer (pink) and a C3 trimer (orange). The O34 octahedral design (bottom panel) was constructed by combining a C3 trimer (pink) and a C4 tetramer (orange). A low-resolution surface representation of each design model is shown (top middle and bottom middle). Simplified two-component cage assemblies with symmetry T (C2 + C3) (top right) and symmetry O (C3 + C4) (bottom right) are diagrammed using model oligomers.

## Protein expression and characterization of co-assembly

Synthetic genes encoding the designed proteins were inserted into expression vectors to allow inducible expression in *Escherichia coli* (see Methods). Each design included a histidine tag on either the N- or the C-terminus of one of the two protein subunits. For six of the designs, either one or both of the engineered protein components were not observed in the soluble fraction of the clarified cell lysate on SDS-PAGE. For these cases, low solubility likely arises from non-specific aggregation or misfolding caused by the introduction of leucine rich coiled coil sequences.

The four remaining designed protein pairs (ccT23-1, ccO34-1, ccO34-3 and ccO23-1) expressed solubly and co-eluted during nickel affinity chromatography and size exclusion chromatography (SEC) (Fig 3A, Fig S5 top, Fig S6 left and middle). Consistent with the target architectures, SEC elution volumes were within the range expected for nanoscale assemblies. Furthermore, negative stain electron microscopy (EM) of the SEC purified samples revealed cage-like species in all four cases (Fig 3B, Fig S5 bottom, Fig S6 right).

Despite these initial results, further structural characterization of ccO23-1 and ccO34-3 was intractable. The trimeric component of the ccO23-1 design exhibited low levels of soluble expression and therefore it proved to be infeasible to produce the two-component assembly for subsequent structural studies. Similarly, only a minute fraction of the ccO34-3 sample eluted at the volume expected for the 48-subunit assembly suggesting that the target design may have only formed as a minor species in solution. This, in addition to issues with sample stability and aggregation, hindered downstream biophysical analysis of ccO34-3. In contrast, with higher yields and dominant SEC peaks in the expected regions, ccT23-1 and ccO34-1 showed more promise. We therefore directed our attention to further characterize the structures of these two designs.

## Crystal structure of a collapsed protein cage

The ccT23-1 design is a ~20 nm cage with tetrahedral symmetry constructed by combining a C2 dimer and a C3 trimer (Fig 1 top). Its dimeric component was derived from a putative isomerase of the SnoaL-like family (PDB ID 3DXO, 18.3 kDa monomeric molecular weight) while its trimeric component was derived from the *Pyrococcus horikoshii* OT3 PH0671 protein (PDB ID

1WY1, 19.5 kDa monomeric molecular weight). When combined with the coiled-coil components, 12 copies of the dimeric subunit and 12 copies of the trimeric subunit would give a total molecular weight of 444 kDa.

Crystals were obtained by hanging drop vapor diffusion after 1-2 weeks, and an X-ray diffraction data set was collected with a resolution limit of 4.32 Å (see Methods). Despite the limited diffraction resolution, because the protein components comprising the assembly were all separately known in advance, the structure could be determined by molecular replacement. Based on the cubic space group of the crystals, we initially expected to find the designed tetrahedral cage sitting at a point of symmetry T in the crystal. However, we determined that the space group was in fact P4(3)32, which lacks any site with T symmetry, and only one copy of the a-b heteromeric design could fit in the asymmetric unit. Ultimately, we established that the structure crystallized as an assembly obeying D3 symmetry, sitting at a site in the crystal conforming to that symmetry. Again, cognizant of the limited diffraction resolution, we used omit maps to validate the resulting structure. The entire alpha helical segment between the trimeric component and the coiled-coil was omitted from phasing, and appeared clearly in an omit map (Fig S1).

Rather than the tetrahedron composed of 12 copies of each subunit that we had designed, the crystal structure obtained from this data revealed a collapsed version of the designed structure, in which only 6 copies of each subunit ($a_6b_6$) instead formed a 222 kDa assembly obeying D3 symmetry (Fig. 2). Instead of the dimeric and trimeric symmetry axes intersecting at an angle of 54.7°, the coiled coil linker was bent significantly such that the axes intersected at 90° (Fig 2A). The collapsed assembly (renamed ccD3 to accurately reflect its symmetry) resembles a nearly rod-shaped structure roughly 20.6 nm long and ~10.4 nm wide (Fig 2B and 2C).

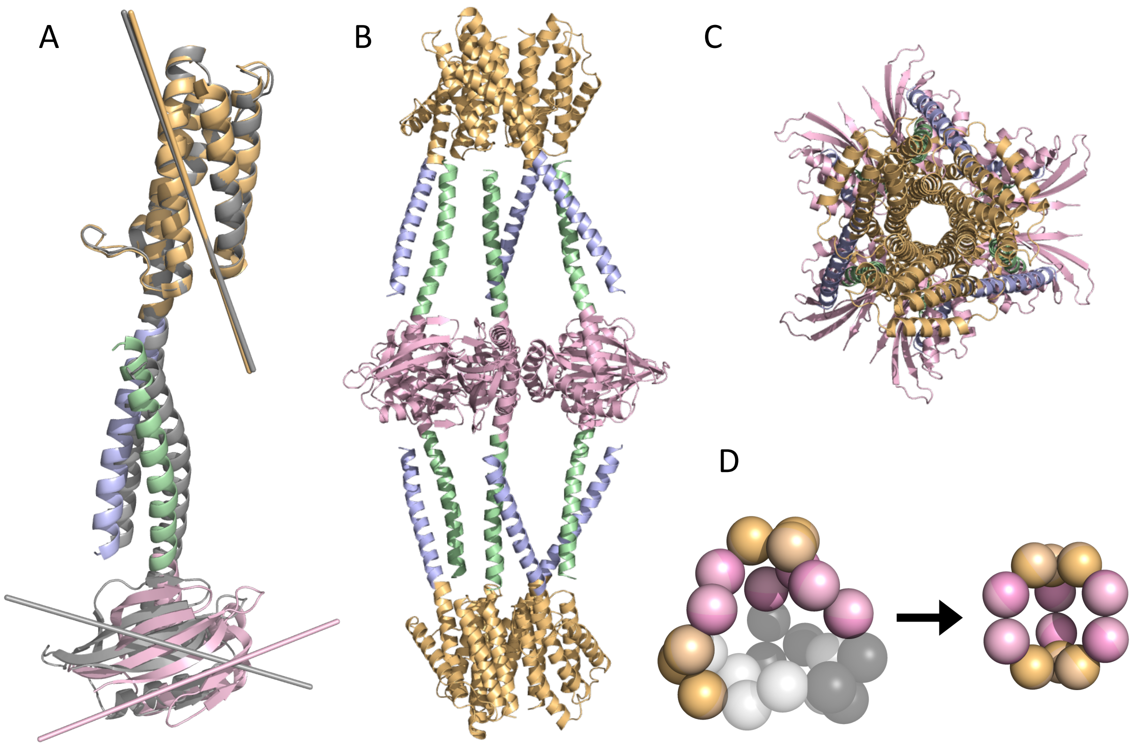


**Figure 2.** Crystal structure of the collapsed T23 design. A) The α-helices joining the dimeric (pink) and trimeric (orange) subunits in the intended design (grey) are bent in the observed crystal structure (colored), with the symmetry axes of the dimer and trimer shown. The observed crystal structure is an assembly with D3 symmetry, viewed from the side (panel B) and down its three-fold symmetry axis (panel C). D) Diagram showing how flexibility can allow a partial assembly intended to be tetrahedral to collapse into a D3 structure. Subunits in the tetrahedral assembly missing from the dihedral assembly are colored in two shades of grey.

Although the crystal structure that was obtained for this design revealed an unintended D3 assembly, a very broad peak in the size exclusion profile that was obtained for the protein in solution suggested a mixture of multiple different assembly species of varying molecular weights (Fig S5A). Indeed, under negative stain EM, we observed particles of different sizes in the sample. 2D classification of manually picked particles revealed a major species with a roughly 21-nm diameter, which corresponds closely to the designed tetrahedral cage (Fig S5C). Although heterogeneity and low particle density prevented us from producing 2D classes that show distinct symmetry elements, many individual particles appear to contain approximate 3-fold symmetry (Fig S5B). Importantly, particle species of higher and lower diameter are also present in the sample; these likely correspond to alternate assembly forms based on different symmetries, such as D3 (as seen in the crystal structure) or even O. Thus, while this cage design case produced a combination of assembled species, apparently including the intended species visible by EM, crystallization selectively favored a smaller, abundant and possibly better ordered form.

**Characterization of a protein octahedron**

The ccO34-1 design is a ~26 nm-wide octahedron with a large interior cavity ~10 nm in diameter (Fig 1 bottom). Its protein components were derived from a putrescine carbamoyltransferase trimer (PDB ID 4AM8, 40 kDa monomer) and a TraM tetramer (PDB ID 2G7O, 8 kDa monomer. When combined with the coiled-coil components, 24 copies of the trimeric subunit plus 24 copies of the tetrameric subunit would give a total molecular weight of 1.29 MDa. The size of the openings or 'windows' joining the interior and exterior space has been characterized for designed and natural protein cages[3]; for ccO34-1 the openings are around ∼30Å in the narrowest direction.

Particles of roughly 25 nm in size were readily observed under negative stain EM (Fig 3B), and reference-free 2D class averages revealed several templates corresponding to views down the 3-fold and 4-fold axes of an octahedral assembly (Fig 3C). Despite these promising preliminary results, the ccO34-1 cage proved resistant to freezing attempts for cryo-EM grid preparation, and crystallization trials did not yield protein crystals suitable for X-ray diffraction experiments. Still, deeper analysis of the cage particles on negatively stained grids provided strong corroborating evidence for formation of the intended octahedral cage, albeit with a degree of flexibility likely responsible for preventing crystal formation.

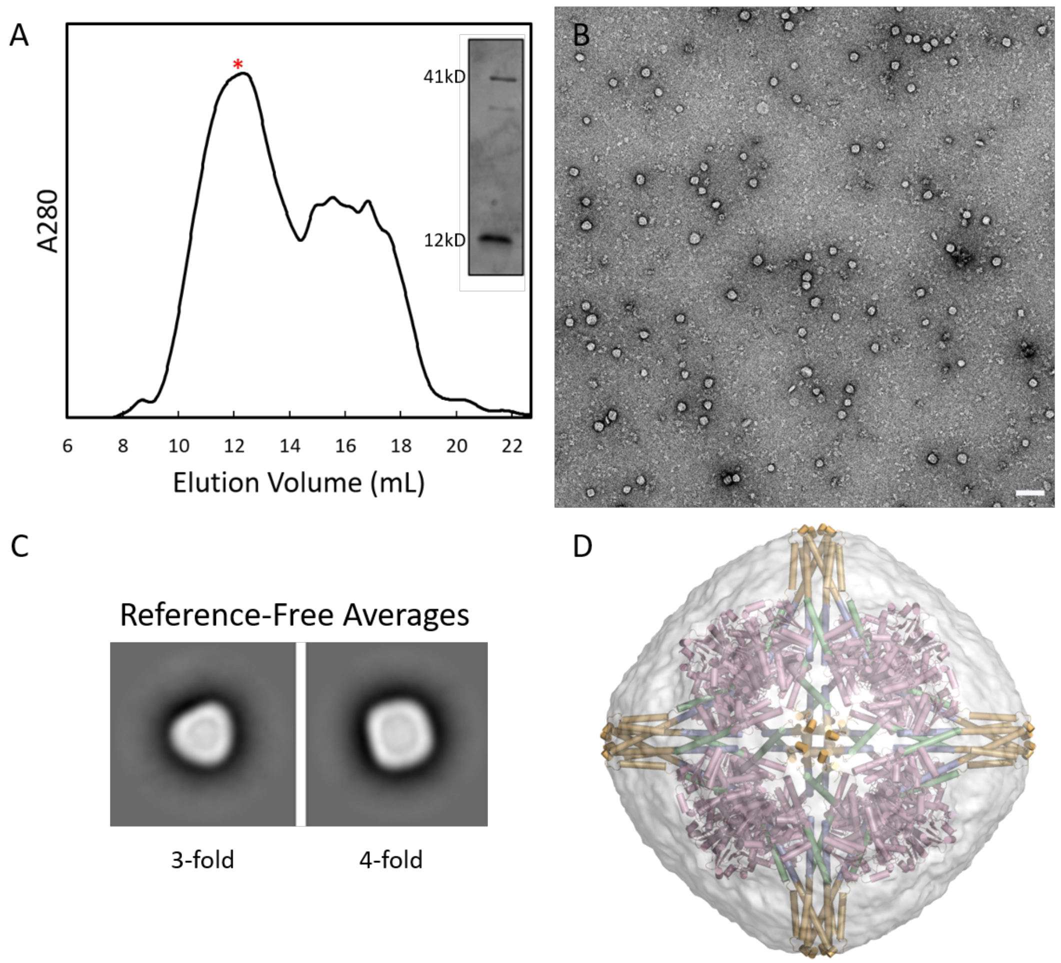


**Figure 3.** Characterization of the ccO34-1 octahedral cage. (A) Purification by size exclusion chromatography produced a major peak (red asterisk) corresponding to a high-molecular weight oligomeric species, comprising both protein cage components, as confirmed by SDS-PAGE (inset). (B) Representative negative stain EM micrograph of the ccO34-1 construct. Scale bar = 100 nm. (C) Selected reference-free class averages of ccO34-1 particles, showing distinct views down the 3-fold and 4-fold axes of the octahedral point group symmetry. (D) The design model of the ccO34-1 cage fit into a 3D reconstruction produced by homogeneous refinement from the negatively stained particles.

From a negative stained EM dataset containing ~3500 particles, 3D reconstruction of a low-resolution electron density map of the structure was pursued. An ab initio 3D reconstruction from these particles performed in cryoSPARC[48] suggested an approximately cubic structure, but with some deviations from perfect octahedral symmetry (Fig S3). Starting from a version of the design model containing only the trimeric component (i.e. with the tetramer and coiled-coil domains removed) low-pass Fourier-filtered to 20 Å resolution, homogeneous refinement was performed in lower symmetries, D2, D4, and T. The omission of a major component from the reference model and the imposition of lower-than-expected symmetry served to validate the robustness of the reconstruction (Fig S3). In all cases, the reconstruction that was obtained clearly resembled an octahedron (i.e. the higher symmetry O emerged without being imposed), fitting the dimensions of the design model, and with density appearing for the omitted tetrameric component. The reconstruction obtained with T symmetry imposed was then further refined by imposing O symmetry (Fig 3D).

Although structural information for the ccO34-1 cage lacked atomic resolution, the data still indicate successful octahedral cage formation. In contrast to the ccT23-1 design, which was shown to favor the unintended ccD3 assembly when crystallized, there was no significant evidence of lower-symmetry assembly species in ccO34-1 samples, despite the appearance of some irregular species and larger aggregates in negative stain EM images (Fig 3B). Differences between the design characteristics of the two cages that may have contributed to these contrasting results are discussed below.

## DISCUSSION AND CONCLUSIONS

Experimental characterizations of the two protein cage designs presented above, along with data from other recent design studies, suggest features of design types that tend to form unanticipated structures[11,23]. Certain combinations of protein oligomers can come together in alternative orientations to produce distinct assemblies having different point group symmetries. This promiscuity naturally permits alternate experimental outcomes. For example, the combination of a C2 dimer and a C3 trimer is particularly promiscuous. Even when restricted to finite structures, this combination can lead to assemblies of I, O, T, or even D3 symmetries, depending on the relative orientation of the components (Fig S7). Indeed, we observed heterogeneous assembly for our C2 plus C3 construction here, despite attempts to limit flexibility. This result parallels that of a previously reported designed protein cube, which was also comprised by a dimer plus trimer combination, held together by a single-helix linker.[23] Although a crystal structure ultimately validated the cubic design in that case, supporting experiments indicated that smaller (i.e. T- and D3-symmetric assemblies) were also obtained in abundance. In contrast, the C3 plus C4 combination is one that does not allow other point symmetry combinations besides O, and indeed our design of this type led to a successful characterization at the level of negative stain EM, and did not produce smaller species in abundance. The favorable designability for cases where alternate symmetric outcomes are absent has been emphasized in other studies[10–12].

A second geometric feature that appears to affect designability relates to the network properties of different kinds of architectures. The potential geometric difficulty of a design could be affected by the "ring size" in a network diagram describing the assembled structure. Both of the designs results in the present study weigh in on this point. For the ccT23-1 design, the C2-

and C3-symmetric oligomers form a triangular ring (or graph cycle) containing 3 subunits of each type – hence, the ring size is 3 (Fig 1, top right). Although it was hypothesized that the hetero coiled coil fusion design method explored here would lead to rigidly oriented component oligomers, experimental studies showed heterogeneity, and ultimately a crystal structured showed preferential formation of an alternate form. In the collapsed D3 version of the assembly obtained by crystallization, the ring formed by the component oligomers is made up of only 2 subunits of each type (ring size of 2) (Fig 2D). In essence, the intended design had ring size 3, but the crystal structure revealed a well ordered alternate assembly with a smaller ring size of 2. For our C3 plus C4 octahedral construction, the ring size for this intended architecture is 2 (the lowest possible value for a cage), and as noted above this case assembled with good robustness and did not collapse into smaller architectures. Another special case published recently further supports these ideas concerning design robustness. A genetic fusion of a pentamer and a dimer did not give robust assembly of an icosahedron; the ring size for that network would have been 3. However, favorable assembly was obtained by fusing three oligomeric components (C2, C3, and C5).[24] In this case, only icosahedral symmetry is jointly compatible with all the component symmetries, and a timer plus pentamer combination gives an icosahedral network with ring size 2. Because self-assembling structures always proceed through intermediate forms, the possibility for incomplete structures to collapse – forming smaller network rings than intended – may be an important general consideration for design. Thus, in order to maximize the chances of design success with somewhat flexible linkages (intentionally or otherwise) between component oligomers, the targeted architecture should be one where the component symmetry types cannot give rise to alternate outcomes with smaller ring sizes.

In this study we explored a new strategy for designing novel symmetric protein cages, exploiting key advantages evident in recent methods. A single alpha helical linker used in earlier work was replaced by a coiled-coil with the goal of reducing flexibility. A heteromeric coil-coil allowed the connection of two oligomeric components by a robust and predictable non-covalent interface. Ten computational designs were tested experimentally, with differing degrees of success leading to informative results. A tetrahedral design, based on a dimeric plus trimeric symmetry combination was found experimentally to be geometrically promiscuous, forming heterogenous assemblies. Size exclusion experiments and negative stain EM analysis suggest that the mixture likely included structures similar to what was designed, but ultimately a prominent component crystallized and was found to represent a smaller, collapsed structure with D3 symmetry ($a_6b_6$ stoichiometry instead of $a_{12}b_{12}$). A second design based on a non-promiscuous combination of trimer and cyclic tetramer to give symmetry O assembled largely as intended. Flexibility was still evident, limiting high resolution by electron microscopy, but lower resolution negative stain imaging confirmed the essential design. Experimental trials such as these provide important guidance for optimizing protein design strategies. Given the overall experimental success rates for designing novel protein assemblies, which are currently rather modest, the heteromeric linker approach described here could be a productive new avenue, especially when applied to favorable symmetry combinations.

## METHODS

### Protein Construct Design

The coiled-coil fusion designs were carried out using a similar approach to the previously described single-helix oligomer fusion method[15,20,23,24] with the primary distinction that the original α-helical linker was substituted by a heterodimeric coiled-coil linker[2] based on c-Fos (coiled-coil segment A) and c-Jun (coiled-coil segment B) (supplemental text). Briefly, homo-oligomeric biological assemblies with C2, C3 and C4 symmetry were downloaded from the Protein Data Bank, and Stride[49] was used to analyze their secondary structure. Then, all possible pairs of dimers and trimers or trimers and tetramers with α-helical termini underwent a multi-step alignment procedure. First, the N-terminus of coiled-coil segment A was used to align the entire heterodimeric Fos/Jun linker to the C-terminal helix of a subunit from oligomer 1. Then, a subunit from oligomer 2 was aligned at its N-terminal α-helix to the C-terminus of coiled-coil segment B. Additionally, helical extensions ranging from 1 to 4 residues were applied to the N- and C-terminus of coiled-coil segments A and B respectively and all possible coiled-coil linker variations were tested for each pair. For each such geometric construction, the angle and distance between the cyclic symmetry axes of the two oligomers were then computed. Alignments that resulted in a distance of 3 Å or less between symmetry axes and an angle within 5° of 54.7° were considered for further analysis. Candidates were then manually inspected, and any that contained major steric clashes were removed. Those exhibiting one or two minor sidechain clashes caused by a helical region in the native sequence of an oligomeric building block were retained. In such cases, clashes were mitigated by alanine point mutations. Among the remaining candidates, 10 designs were selected for experimental testing.

## Synthetic Gene Construction

*E. coli* codon optimized gene fragments encoding the different protein design constructs were purchased from Integrated DNA Technologies. Sequences encoding the dimer and trimer subunits of the ccT23-1 design were inserted into the pCDFDuet-1 and pET-22b expression vectors (Novagen) respectively *via* Gibson Assembly. Sequences encoding the trimer and tetramer subunits of the ccO34-1 design were both inserted into the same pET-22b plasmid separated by the intergenic region of pETDuet-1 (Novagen).

## Protein Expression and Purification

Both the ccT23-1 dimer and trimer subunits were transformed separately into *E. coli* BL21 (DE3) cells (New England Biolabs). Starter cultures were grown in LB with overnight shaking at 37° C and used to inoculate 1 L of LB media supplemented with spectinomycin (pCDFDuet-1) or ampicillin (pET-22b) at 75 μg/mL. Cultures were grown with shaking at 37°C until reaching an $OD_{600}$ of 0.6, after which protein expression was induced with 0.4 mM IPTG. Cells were incubated for a further 4 hours with shaking at 37°C before being harvested by centrifugation at 5000 x g for 10 min. Harvested cells were suspended in lysis buffer (50 mM Tris pH 7.5, 500 mM NaCl, 2.8 mM β-mercaptoethanol, 5 mM $MgCl_2$ and 10% Glycerol) supplemented with 10mM imidazole pH 7.5 and EDTA free protease inhibitor (Pierce; Thermo Fisher Scientific) and co-lysed with an Emulsiflex C3 (Avestin). Lysate was clarified by centrifugation at 9500 x g for 40min at 4°C. For nickel-affinity purification, the soluble lysate fraction was incubated at 4°C for 90 min with 4 mL of Ni-NTA resin on a rocker before being loaded onto a gravity column and washed with 100 mL of lysis buffer containing 75mM imidazole. The protein was eluted from the column with 20 mL of lysis buffer containing 250mM imidazole. The eluted sample was concentrated to 4 mg/mL, then further purified by size exclusion chromatography (SEC) on a Superose 6 30/100 column (GE Healthcare) equilibrated with SEC buffer (0.5M NaCl, 5% Glycerol, 2.8mM β-mercaptoethanol, 50 mM Tris pH 7.5 and 5mM $MgCl_2$). Elution fractions immediately after the void volume were evaluated by SDS-PAGE. Fractions containing both ccT23 subunits were pooled for further characterization.

*E. coli* BL21 (DE3) cells (New England Biolabs) cells transformed with the ccO34-1 expression plasmid were grown in P-0.5G non-inducing minimal medium overnight.[50] 10 mL of cell culture was then used to inoculate 1L of ZYP-5052 auto-inducing medium supplemented with ampicillin at 75 µg/mL for 65 hours at 20°C (Studier 2005). Cells were harvested by centrifugation at 4000 x g for 20 minutes, resuspended in lysis buffer (50mM HEPES pH 7.0, 5% glycerol, 250mM NaCl and 2.8mM β-mercaptoethanol) supplemented with EDTA free protease inhibitor (Pierce; Thermo Fisher Scientific) and lysed with an Emulsiflex C3 (Avestin). Lysate was centrifuged at 12,000 x g for 35 minutes at 4°C and the soluble fraction was applied to a 4 mL Ni-NTA gravity column. A series of 25mL wash buffers (lysis buffer with 25 mM, 40 mM or 75 mM imidazole) were applied to the column sequentially with increasing amounts of imidazole before eluting the target protein with lysis buffer containing 250 mM imidazole. The eluate was dialyzed overnight into 25 mM HEPES pH 7.0, 250 mM NaCl, 2% Glycerol and 1.4mM β-mercaptoethanol at 4°C. The dialyzed protein was then further purified by SEC on a Superose 6 30/100 column (GE Healthcare) equilibrated with dialysis buffer. Elution fractions were evaluated by SDS-PAGE, and fractions containing both ccO34 subunits were pooled for further analysis.

**Protein Crystallization**

The ccT23-1 protein sample was concentrated to 2.8 mg/mL, passed through a 0.22 µm filter and centrifuged at 9500 x g for 10min at 4° C. The target protein was crystallized using the hanging-drop vapor diffusion method. Crystallization trials were performed at the UCLA Crystallization Facility in 96-well trays set with 210 nL drops using a Mosquito liquid handling device (TPP LabTech). The crystal used for structure determination was obtained using the Silver Bullets additive screen by Hampton Research. The reservoir contained 90 µl of crystallization reagent (0.10 M Sodium Acetate pH 5.4, 66% MPD) and 10 µl of Silver Bullets reagent B9 (0.25% w/v Hexamminecobalt(III) chloride, 0.25% w/v Salicylamide, 0.25% w/v Sulfanilamide, 0.25% w/v Vanillic acid, 0.02 M HEPES sodium pH 6.8). The hanging drop contained purified protein and reservoir solution in a 2:1 (v/v) ratio. Crystals took about 7 to 14 days to grow at 4° C.

**X-ray Diffraction Data Collection and Structure Determination**

X-ray diffraction data from a single ccD3 crystal were collected at beamline 24-ID-C of the Advanced Photon Source, Argonne National Laboratory, Argonne, IL, USA, at a wavelength of 0.9790 Å and temperature of 100 K. Data were collected using 0.5° oscillations and 697 mm detector distance with a DECTRIS PILATUS-6MF pixel detector. Indexing and integration of the reflections were performed using XDS in space group $P4_332$ and scaled with XSCALE to a resolution of 4.32 Å.[51] Initial molecular replacement calculations conducted in cubic space groups appeared unreliable, so the structure was solved by molecular replacement in space group P1 using the program Phaser[52] with search models 3DXO and 1WY1 corresponding to the dimer and trimer components of our ccT23 design respectively. A difference density map revealed positions of the coiled-coils which were then modeled using Coot.[53] We manually rotated and translated the coiled-coils from 4G1E to fit the density and made adjustments to the dimer and trimer regions. This model was refined using Phenix.[54] A single heterodimer from the improved model was retrospectively used as a search model for molecular replacement in cubic space groups $P4_132$ and $P4_332$; the correct solution was evident in the latter. We proceeded with rigid body refinement and then with simulated annealing and gradient-driven minimization. Due to the limited resolution of the dataset we applied multiple geometric restraints including rotamer restraints, hydrogen bond restraints for the coiled-coil region and restraints to reference models 3DXO and 1WY1. Furthermore, atomic displacement parameters were grouped throughout the refinement process and anisotropic displacement was modeled using a single TLS group. We alternated between Phenix refinement and manual building for several iterations. Data collection and refinement statistics are reported in Table S1.

**Glutaraldehyde Crosslinking of ccT23**

After nickel-affinity purification the ccT23 sample was further purified by size exclusion chromatography on a Superose 6 30/100 column (GE Healthcare) equilibrated with 500mM NaCl, 20mM HEPES pH 7.5, 5mM $MgCl_2$, 10% Glycerol and 1.4 mM β-mercaptoethanol. Elution fractions immediately after the void volume were pooled and concentrated to 1.1 mg/mL. The purified sample was crosslinked using 0.02% glutaraldehyde (Ted Pella). The reaction was quenched after 4 minutes with 100mM Tris pH 8. The crosslinked sample was diluted to either

0.01mg/ml or 0.35mg/ml in 300mM NaCl, 15mM Tris pH 7.5, 5mM $MgCl_2$, 2% Glycerol and 1.4 mM beta-mercaptoethanol for analysis by negative stain electron microscopy.

**Negative Stain Electron Microscopy**

Protein samples were applied onto a glow discharged, 300-mesh carbon-coated formvar-supported copper grid (Ted Pella), washed with Milli-Q water and stained with 2% uranyl acetate. Initial sample screening was performed on a Tecnai T12 transmission electron microscope. Micrographs used for further structural characterization of ccO34-1 and ccT23-1 were collected on an FEI TF20 transmission electron microscope at 29,000X and 50,000X magnification respectively.

Electron microscopy data for the ccO34-1 construct were processed using cryoSPARC 2.15.0.[48] Reference free 2D class averages were obtained from 7,257 auto-picked particles. Homogeneous refinement was performed on the particles using a reference map consisting of a 20 Å-filtered version of the design model with the tetrameric component and coiled-coil linker removed. For the final 3D reconstruction, a first round of homogeneous refinement was performed with T symmetry imposed, followed by subsequent rounds which imposed O symmetry.

Data for the ccT23-1 construct were processed using cryoSPARC and Relion 3.1.[55] 2,727 particles were picked manually and extracted in cryoSPARC. Reference-free 2D class averages were produced in Relion.

**Acknowledgments**

The authors thank Duilio Cascio and Michael Collazo for help with crystallization trials as well as Prof. Hong Zhou and Ivo Atanasov for guidance on electron microscopy. This work was supported by NSF grant CHE-1629214. This work used NE-CAT beamline 24-ID-C (GM124165) and a Pilatus detector (RR029205) at the APS (DE-AC02-06CH11357). We thank the NE-CAT staff for their assistance with data collection.

**Notes**

The authors declare no conflicting interests.

**Data Deposition**

Atomic coordinates and structure factors were deposited in the Protein Data Bank under the accession code 6X1I.

**Author Contributions**

The work was conceived by TOY. Protein design was done by JL and DEM. Cloning was done by JL and JEM with some help from DEM. Protein expression and purification was done by JL and JEM. Crystallization and EM data collection was performed by JL. Negative stain EM 3D reconstruction and 2D class averages were carried out by KAC. X-Ray crystal structure determination was performed by MRS and JL. The written manuscript was prepared by KAC, TOY and JL.

**Supplementary Information for:**

Geometric Lessons and Design Strategies for Nanoscale Protein Cages

Joshua Laniado, Kevin A. Cannon, Justin E. Miller, Michael R. Sawaya, Dan E. McNamara and Todd O. Yeates

**Contents:**

- 2 Supplementary Tables
- 7 Supplementary Figures
- Supplementary Text

## SUPPLEMENTAL TABLES

Table S1. **Crystallographic table**

| | |
|---|---|
| Space group | $P4_332$ |
| Cell dimensions a, b, c (Å) | 146.34, 146.34, 146.34 |
| Resolution (Å) | 84.5 – 4.32 |
| % Data completeness | 99.5 (96.9) |
| Data redundancy | 17.7 (16.8) |
| $R_{merge}$ | 0.10 (2.37) |
| I/σ(I) | 14.40 (1.07) |
| CC ½ % | 100.0 (41.1) |
| $R_{work}$ | 0.278 (0.404) |
| $R_{free}$ | 0.293 (0.425) |
| Ramachandran (Favored, Allowed, Outliers) | 97.8%, 2.2%, 0 % |
| Average B value Chain A (1354 atoms) | 258 Å$^2$ |
| Average B value Chain B (1153 atoms) | 279 Å$^2$ |

Parenthetic values refer to the outer resolution shell.

Table S2. **Amino Acid Sequences**

| | |
|---|---|
| ccT23-1 Dimer (based on 3DXO) | MAQLKKKLQALKKKNAQLKWKLQALKKKLAQATQHLTIAQTYLAAWNEEDNERRRHLVGQAWAENTRYVDPLMQGEGQQGIAAMIEAARQKFPGYRFVLAGTPDGHGNFTRFSWRLISPDGDDVAGGTDVVSLNTEGRIDNVVGFLDGAVSHHHHHH |
| ccT23-1 Trimer (based on 1WY1) | MSPIIEANGTLDELTSFIGEAKHYVDEEMKGILEEIQNDIYKIMGEIGSKGKIEGISEERIKWLEGLISRYEEMVNLKSFVLPGGTLESAKLDVCRTIARRAERKVATVLREFGIGKEALVYLNRLSDLLFLLARVIEIAAAAQLEKELQALEKENAQLEWELQALEKELAQ |
| ccO34-1 Trimer (based on 4AM8) | MKRDYVTTETYTKEEMHYLVDLSLKIKEAIKNGYYPQLLKNKSLGMIFQQSSTGTRVSFETAMEQLGGHGEYLAPGQIQLGGHETIEDTSRVLSRLVDILMARVERHHSIVDLANCATIPVINGMSDYNHPTQELGDLCTMVEHLPEGKKLEDCKVVFVGDATQVCFSLGLITTKMGMNFVHFGPEGFQLNEEHQAKLAKNCEVSGGSFLVTDDASSVEGADFLYTDVWYGLYEAELSEEERMKVFYPKYQVNQEMMDRAGANCKFMHCLPATRGEEVTDEVIDGKNSICFDEAENRLTSIRGLLVYLMNDYEAKNPYDLIKQAAAKKALEVFLDTQAAAAQLEKELQALEKENAQLEWELQALEKELAQ |
| ccO34-1 Tetramer (based on 2G7O) | MAQLKKKLQALKKKNAQLKWKLQALKKKLAQAAQTEFNKLLLECVVKTQSSVAKILGIESLSPHVSGNSKFEYANMVEDIREKVSSEMERFFPKNDDEHHHHHH |
| ccO34-2 Trimer (based on 2V82) | MHHHHHHQWQTKLPLIAILRGITPDEALAHVGAVIDAGFDAVEIPLNSPQWEQSIPAIVDAYGDKALIGAGTVLKPEQVDALARMGCQLIVTPNIHSEVIRRAVGYGMTVCPGCATATEAFTALEAGAQALKIFPSSAFGPQYIKALKAVLPSDIAVFAVGGVTPENLAQWIDAGCAGAGLGSDLYRAGQSVERTAQQAAAFVKAYREAVQAQLEKELQALEKENAQLEWELQALEKELAQ |
| ccO34-2 Tetramer (based on 1E4C) | MAQLKKKLQALKKKNAQLKWKLQALKKKLAQAAARNKLARQIIDTCLEMTRLGLNQGTAGNVSVRYADGMLITPTGIPYEKLTESHIVFIDGNGKHEEGKLPQSEWRFHMAAYQSRPDANAVVHNHAVHCTAVSILNRSIPAIHYMIAAAGGNSIPCAPYATFGTRELSEHVALALKNRKATLLQHHGLIACEVNLEKALWLAHEVEVLAQLYLTTLAITDPVPVLSDEEIAVVLEKFKTFGLRIEE |
| ccO34-3 Trimer (based on 4G9Q) | MHHHHHHSSGVDLGTENLYFQSMMTTSNAGAQQPNVEGRRFSPDQVRSVAPALEQYTQQRLYGDVWQRPGLNRRDRSLVTIAALIARGEAPALTYYADQALENGVKPSEISETITHLAYYSGWGKAMATVGPVSEAFAKRGIGQDQLAAVESTPLPLDEEAEAQRATTVGNQFGSVAPGLVQYTTDYLFRDLWLRPDLAPRDRSLVTIAALISVGQVEQITFHLNKALDNGLSEEQAAEVITHLAFYAGWPNAMSALPVAKAVFEKRRAAQLEKELQALEKENAQLEWELQALEKELAQ |
| ccO34-3 Tetramer (based on 1KBJ) | MAQLKKKLQALKKKNAQLKWKLQALKKKLAQAAAKEDIARKEQLKSLLPPLDNIINLYDFEYLASQTLTKQAWAYYSSGANDEVTHRENHNAYHRIFFKPKILVDVRKVDISTDMLGSHVDVPFYVSATALCKLGNPLEGEKDVARGCGQGVTKVPQMISTLASCSPEEIIEAAPSDKQIQWYQLYVNSDRKITDDLVKNVEKLGVKALFVTVDAPSLGQREKDMKLKFSNTKAGPKAMKKTNVEESQGASRALSKFIDPSLTWKDIEELKKKTKLPIVIKGVQRTEDVIKAAEIGVSGVVLSNHGGRQLDFSRAPIEVLAETMPILEQRNLKDKLEVFVDGGVRRGTDVLKALCLGAKGVGLGRPFLYANSCYGRNGVEKAIEILRDEIEMSMRLLGVTSIAELKPDLLDLSTLKARTVGVPNDVLYNEVYEGPTLTEFEDA |
| ccO34-4 Trimer (based on 2V82) | MHHHHHHQWQTKLPLIAILRGITPDEALAHVGAVIDAGFDAVEIPLNSPQWEQSIPAIVDAYGDKALIGAGTVLKPEQVDALARMGCQLIVTPNIHSEVIRRAVGYGMTVCPGCATATEAFTALEAGAQALKIFPSSAFGPQYIKALKAVLPSDIAVFAVGGVTPENLAQWIDAGCAGAGLGSDLYRAGQSVERTAQQAAAFVKAYREAVQAAAQLEKELQALEKENAQLEWELQALEKELAQ |

| | |
|---|---|
| ccO34-4 Tetramer (based on 2FLF) | MAQLKKKLQALKKKNAQLKWKLQALKKKLAQAQRAERARLYAAFRQVGEDLFAQGL ISATAGNFSVRTKGGFLITKSGVQKARLTPEDLLEVPLEGPIPEGASVESVVHREVYRRT GARALVHAHPRVAVALSFHLSRLRPLDLEGQHYLKEVPVLAPKTVSATEEAALSVAEAL REHRACLLRGHGAFAVGLKEAPEEALLEAYGLMTTLEESAQILLYHRLWQGAGPALGG GE |
| ccO34-5 Trimer (based on 4IV5) | MLELPPVASLGGKSITSAEQFSRADIYALIHLASAMQRKIDAGEVLNLLQGRIMTPLFFE DSSRTFSSFCAAMIRLGGSVVNFKVEASSINKGETLADTIRTLDSYSDVLVMRHPRQDA IEEALSVAQHPILNAGNGAGEHPTQALLDTLTIHSELGSVDGITIALIGDLKMGRTVHSL LKLLVRNFSIKCVFLVAPDALQMPQDVLEPLQHEIATKGVIIHRTHALTDEVMQKSDVL YTTRLQKERFMASTSDDAAALQSFAAKADITIDAARMRLAKEKMIVMHPLPRNDELS TTVDADPRAAYFRQMRYGMFMRMAILWSVLAAAQLEKELQALEKENAQLEWELQA LEKELAQ |
| ccO34-5 Tetramer (based on 2FLF) | MAQLKKKLQALKKKNAQLKWKLQALKKKLAQAQRAERARLYAAFRQVGEDLFAQGLI SATAGNFSVRTKGGFLITKSGVQKARLTPEDLLEVPLEGPIPEGASVESVVHREVYRRTG ARALVHAHPRVAVALSFHLSRLRPLDLEGQHYLKEVPVLAPKTVSATEEAALSVAEALR EHRACLLRGHGAFAVGLKEAPEEALLEAYGLMTTLEESAQILLYHRLWQGAGPALGG GEHHHHHH |
| ccO34-6 Trimer (based on 3Q1X) | MAQLKKKLQALKKKNAQLKWKLQALKKKLAQAADNYIYSIAHQLYEMYLQDEDAFHS KRDYPHKKVFTELQKLRKIFFPDFFMKHQKITESHIASELTKLVDYIKDSVTAYNDELFAA QCVMAILEKLPSIKRTLKTDLIAAYAGDPAAPGLSLIIRCYPGFQAVIVYRIAHVLYECGE RYYCREMMESVHSYTSIDIHPGASIKGHFFIDHGVGVVIGETAIIGEWCRIYQSVTLGA MHFQEEGGVIKRGTKRHPTVGDYVTIGTGAKVLGNIIVGSHVRIGANCWIDRDVDSN QTVYISEHPTHFVKPCTTKGMKNDTEIIAIIPSSPLANSPSILEHHHHHH |
| ccO34-6 Tetramer (based on 3RPZ) | MSNAMNVPFWTEEHVGATLPERDAESHKGTYGTALLLAGSDDMPGAALLAGLGAM RSGLGKLVIGTSENVIPLIVPVLPEATYWRDGWKKAADAQLEETYRAIAIGPGLPQTES VQQAVDHVLTADCPVILDAGALAKRTYPKREGPVILTPHPGEFFRMTGVPVNELQKK RAEYAKEWAAQLQTVIVLKGNQTVIAFPDGDCWLNPTGNGALAKGGTGDTLTGMIL GMLCCHEDPKHAVLNAVYLHGACAELWTDEHSAHTLLAHELSDILPRVWKRFEAAAA QLEKELQALEKENAQLEWELQALEKELAQ |
| ccO34-7 Trimer (based on 3OER) | MSVESSTDGQVVPQEVLNLPLEKAHEEADDYLDHLLDSLEELSEAHPDCIPDVELSHG VMTLEIPAFGTYVINKQPPNKQIWLASPLSGPNRFDLLNGEWVSLRNGTKLTDILTEEV EKAISEAAAQLEKELQALEKENAQLEWELQALEKELAQ |
| ccO34-7 Tetramer (based on 2Z7B) | MAQLKKKLQALKKKNAQLKWKLQALKKKLAQAQARRKVFEELVTATKILLNEGIMDT FGHISARDPEDPASFFLAQKLAPSLITVDDIQRFNLDGETSDNRPSYLERYIHSEIYKTRP DVQCVLHTHSPAVLPYCFVDTPLRPVTHMGAFIGESVPVYEIRDKHGDETDLFGGSPD VCADIAESLGSQTVVLMARHGVVNVGKSVREVVFRAFYLEQEAAALTAGLKIGNVKYL SPGEIKTAGKLVGAQIDRGWNHWSQRLRQAGLAHHHHHH |
| ccO34-8 Trimer (based on 2P4S) | MYTYDTLQEIATYLLERTELRPKVGIICGSGLGTLAEQLTDVDSFDYETIPHFPVSTVAG HVGRLVFGYLAGVPVMCMQGRFHHYEGYPLAKCAMPVRVMHLIGCTHLIATNAAG GANPKYRVGDIMLIKDHINLMGFAGNNPLQGPNDERFGPRFFGMANTYDPKLNQQ AKVIARQIGIENELREGVYTCLGGPNFETVAEVKMLSMLGVDAIGMSTVHEIITARHC GMTCFAFSLITNMCTMSYEEEEEHCHDSIVGVGKNREKTLGEFVSRIVKHIHYEAAQLE KELQALEKENAQLEWELQALEKELAQ |
| ccO34-8 Tetramer (based on 2R9G) | MAQLKKKLQALKKKNAQLKWKLQALKKKLAQAQHYDVISAFQKSIRGSDVDAALHYL ARLVEAGDLASICRRLMVIGYEDIGLGNPAAAARTVNAVLAAEKLGLPEARIPLADVVV DLCLSPKSNSAYMALDAALADIREGKAGDVPDHLRDSHYKGAKSLNRGVGYQYPHHF DQAWVNQQYLPDKLKNAQYYQPKDTGKYEQALGQQYYRIKEWKEHHHHHH |

| | |
|---|---|
| ccO23-1 Dimer (based on 3OCU) | MAQLKKKLQALKKKNAQLKWKLQALKKKLAQAEHANMQLQQQAVLGLNWMQDS GEYKALAYQAYNAAKVAFDHAKVAKGKKKAVVADLNETMLDNSPYAGWQVQNNK PFDGKDWTRWVDARQSRAVPGAVEFNNYVNSHNGKVFYVTNRKDSTEKSGTIDDM KRLGFNGVEESAFYLKKDKSAKAARFAEIEKQGYEIVLYVGDNLDDFGNTVYGKLNAD RRAFVDQNQGKFGKTFIMLPNANYGGWEGGLAEGYFKKDTQGQIKARLDAVQAW DGKHHHHHH |
| ccO23-1 Trimer (based on 1VL0) | MKILITGANGQLGREIQKQLKGKNVEVIPTDVQDLDITNVLAVNKFFNEKKPNVVINC AAHTAVDKCEEQYDLAYKINAIGPKNLAAAAYSVGAEIVQISTDYVFDGEAKEPITEFDE VNPQSAYGKTKLEGENFVKALNPKYYIVRTAWLYGDGNNFVKTMINLGKTHDELKVV HDQVGTPTSTVDLARVVLKVIDEKNYGTFHCTCKGICSWYDFAVEIFRLTGIDVKVTPC TTEEFPRPAKRPKYSVLRNYMLELTTGDITREWKESLKEYIDLLQMAAAQLEKELQALE KENAQLEWELQALEKELAQ |

## SUPPLEMENTAL FIGURES

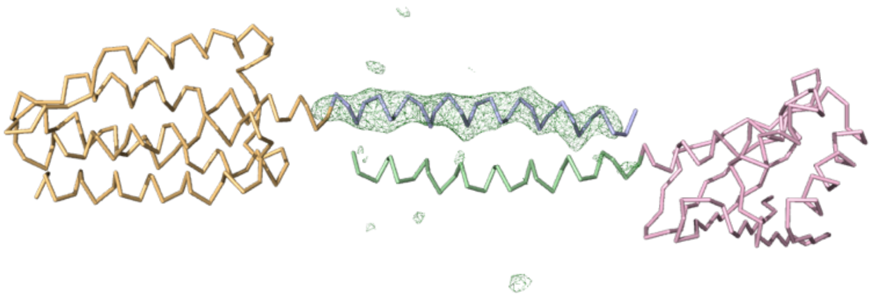


**Figure S1**. Electron density map of the ccD3 structure. An omit map (green mesh) was used to establish the correctness of the molecular replacement solution. The density map, computed after omitting the coiled-coil segment (blue) fused to the trimer subunit (orange), clearly reveals positive density for the omitted region. The dimer subunit is shown in pink with its coiled-coil segment in green.

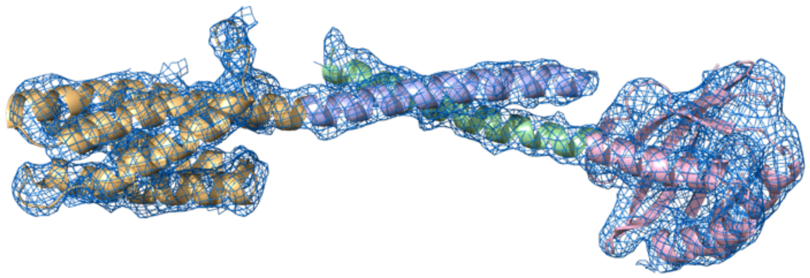


**Figure S2**. Electron density map for the ccD3 crystal structure. A 2Fo-Fc electron density map (blue mesh) contouring the asymmetric unit of the ccD3 crystal structure at 1.1 σ.

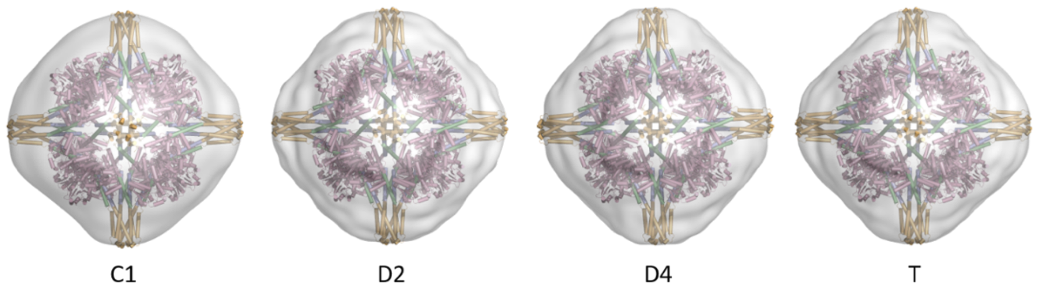


**Figure S3**. 3D single particle electron microscopy reconstructions produced by homogeneous refinement with various types of symmetry imposed for the ccO34-1 octahedral cage. Regardless of the lower symmetry that was imposed, the reconstructions conform to near-octahedral symmetry. The reconstructions were generated using as a reference a low resolution (20 Å) map of a partial model from which major components – the tetramer (colored in yellow) and coiled-coil components – were removed as a test of robustness.

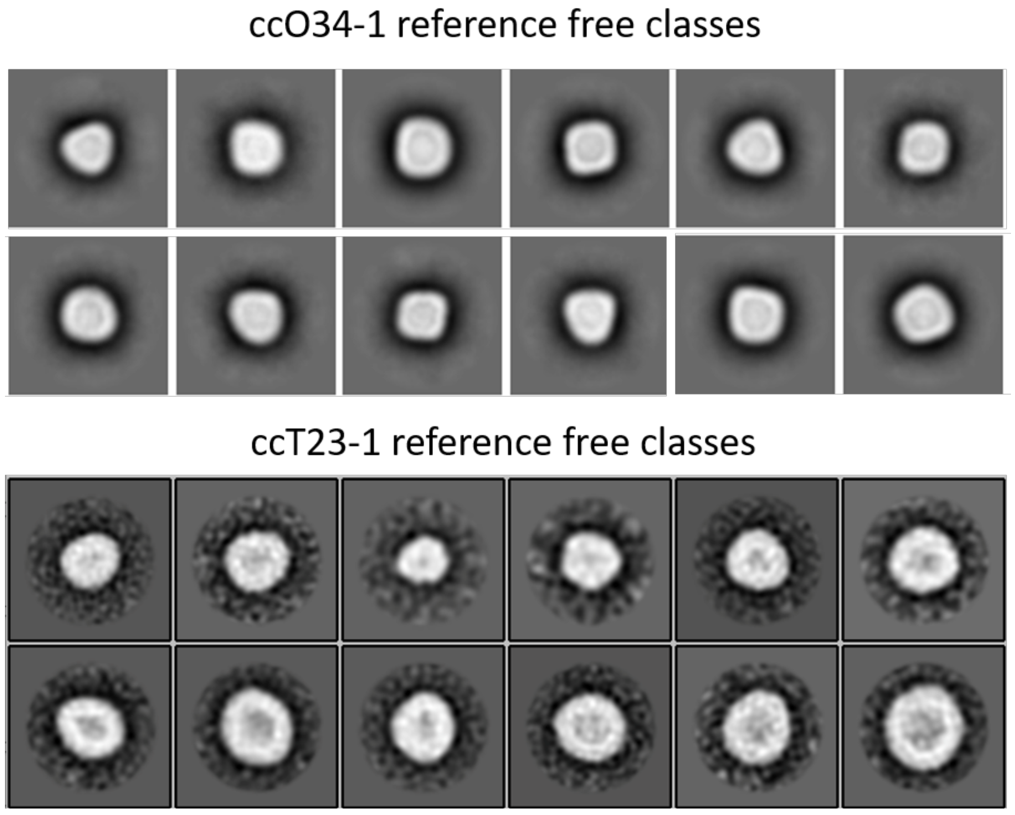

ccO34-1 reference free classes
ccT23-1 reference free classes

**Figure S4.** Reference-free 2D class averages of particles for the ccO34-1 (top) and ccT23-1 (bottom) cages from negative stain EM images. The ccO34-1 construct produced cages of relatively uniform size (roughly 25 nm), whereas the ccT23-1 construct showed particles with a range of sizes. Medium size particles of ccT23-1 are in the range of 20-22 nm, which is close to the correct size for the designed tetrahedron. The smallest class observed is 14 nm in size, which could correspond to smaller species like the D3 assembly which was observed by X-ray crystallography. The largest classes reach about 29 nm in size, which could correspond to assemblies with more subunits, e.g. octahedra with 24 subunits. Classes are ordered from highest to lowest relative abundance (top left to bottom right).

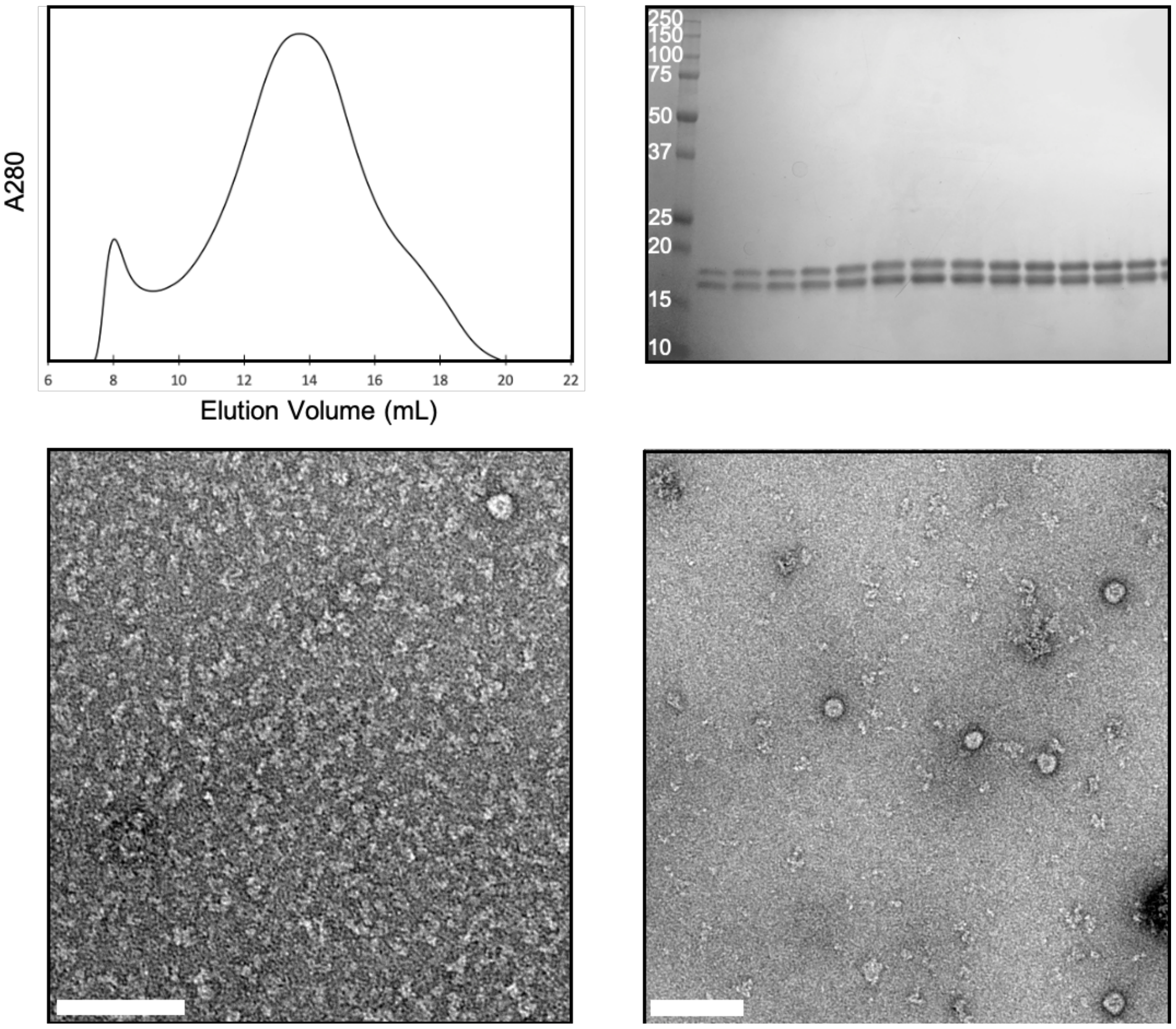


**Figure S5.** Purification and glutaraldehyde crosslinking of ccT23-01. A broad major absorbance peak (11mL-17mL) occurs immediately after the void volume on SEC (top left). Bands for both dimer and trimer subunits appear on SDS-PAGE in all of the 0.5 mL SEC elution fractions corresponding to the peak (top right). The left-most lane of the gel contains the Precision Plus Protein Standard marker (Bio-Rad). Molecular weights are in kDa. A low protein concentration (approx. 0.01 mg/mL) is required to prepare negative stain EM grids of the ccT23-01 sample. At such concentrations, mainly small protein species are observed under negative stain EM and cage-like particles are scarce (bottom left). The ccT23-01 sample was crosslinked with 0.02% glutaraldehyde for 4 minutes at a higher protein concentration (approx. 1.1 mg/mL) and was then diluted to approx. 0.01 mg/mL. Cage-like particles are significantly more abundant in the crosslinked sample under negative stain EM (bottom right). Negative stain micrographs were collected on a Tecnai T12 transmission electron microscope. Scale bars (white) are 100 nm.

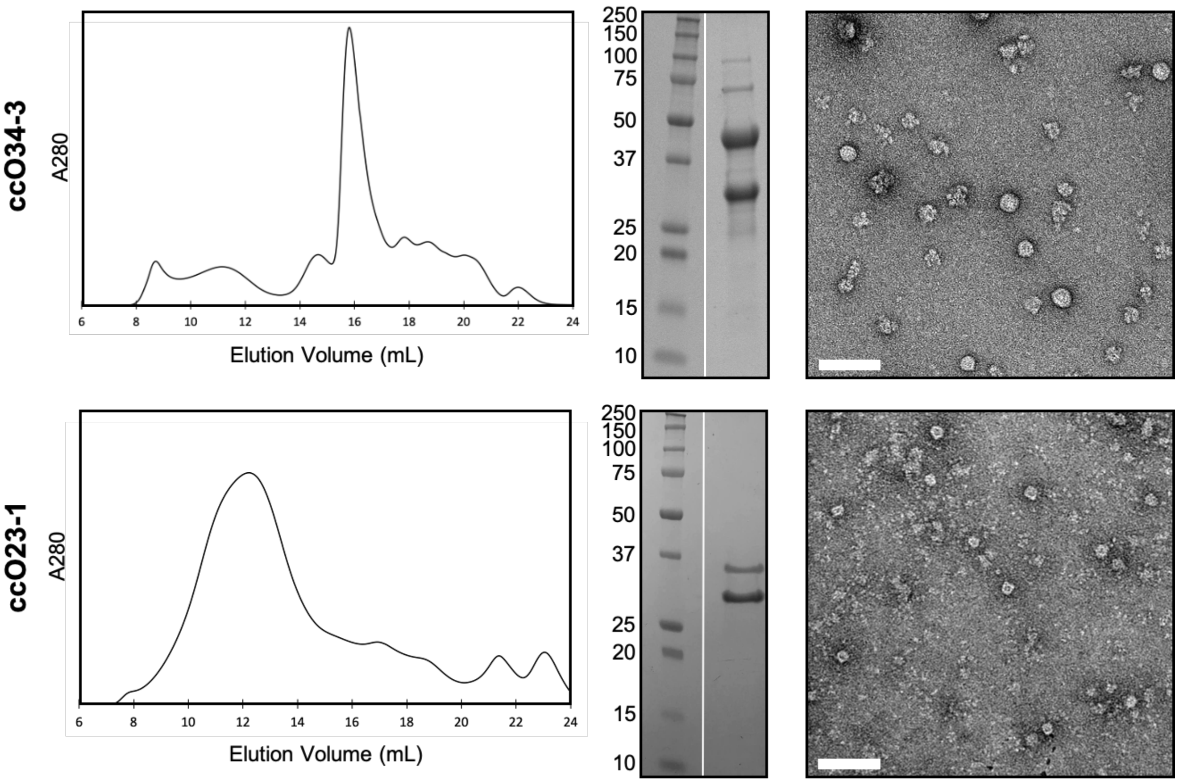


**Figure S6.** Characterization of ccO34-3 and ccO23-1 designs. The ccO34-3 construct exhibits a small yet narrow absorbance peak (10mL-12mL) immediately after the void volume on SEC (top left). Bands for both trimer and tetramer subunits appear on SDS-PAGE of the pooled elution fractions of the SEC peak (top middle). Although protein aggregates and partial assemblies are prominent under negative stain EM, cage-like assemblies are readily observed (top right). The ccO23-1 construct exhibits a broad absorbance peak (9mL-14mL) immediately after the void volume on SEC (bottom left). Bands for both dimer and trimer subunits appear on SDS-PAGE of the pooled elution fractions of the SEC peak (bottom middle). Cage-like assemblies as well as protein aggregates are readily observed under negative stain EM (bottom right). Left SDS-PAGE lanes contain the Precision Plus Protein Standard marker (Bio-Rad). Molecular weights are in kDa. Negative stain micrographs were collected on a Tecnai T12 transmission electron microscope and scale bars (white) are 100 nm.

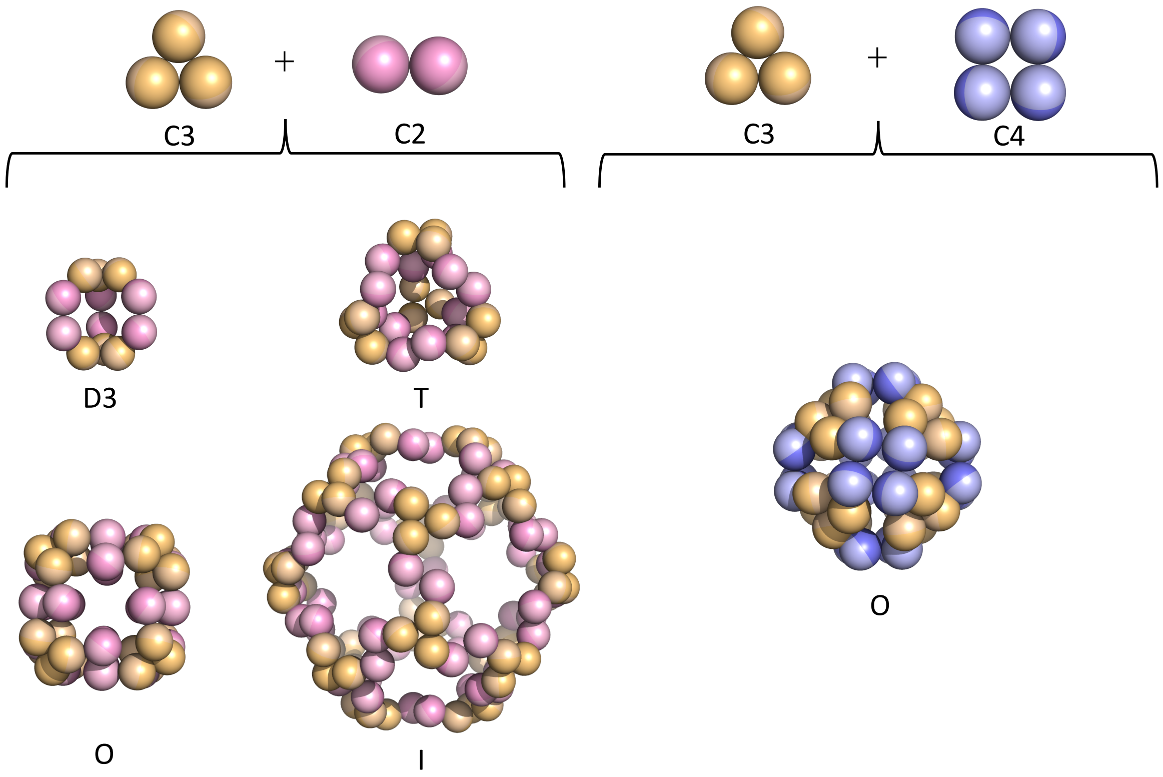


**Figure S7.** Possible finite symmetries from combinations of oligomers in this study. (Left) The combination of a C2 dimer and a C3 trimer can give rise to four different types of finite two-component symmetric assemblies (D3, T, O and I). (Right) The combination of a C3 trimer and C4 tetramer can only give rise to one (O).

## SUPPLEMENTAL TEXT

The model of the heterodimeric coiled-coil helical linker used in this study was generated from the crystal structure of the AP-1 c-Fos-c-Jun transcription factor bound to DNA.[56] The slight bending of the helices as a result of their interaction with DNA was accounted for by replacing the affected N-terminal and C-Terminal helical segments of c-Fos and c-Jun respectively with an idealized model of a 10-residue alanine α-helix. The sequences of c-Fos and c-Jun became AAAAAAAAAANRRRELTDTLQAETDQLEDEKSALQTEIAN and KRKLERIARLEEKVKTLKAQNSELASTANMAAAAAAAAAA respectively. The 10-residue alanine extensions in the resulting model were used to align the coiled-coil linker to the helical termini of symmetric oligomers during the design procedure. Furthermore, we identified sequences of a previously designed coiled-coil based on c-Fos-c-Jun with a significantly higher specificity for heterodimerization over homodimerization.[57,58] Therefore, in order to favor hetero over self-association of identical subunits, before ordering the genes encoding our final designs, we substituted the original c-Fos and c-Jun sequences for their engineered counterparts AQLEKELQALEKENAQLEWELQALEKELAQ (ACID-p1) and AQLKKKLQALKKKNAQLKWKLQALKKKLAQ (BASE-p1) respectively.

## REFERENCES

(1) Huang, P.-S.; Boyken, S. E.; Baker, D. The Coming of Age of *de Novo* Protein Design. *Nature* 2016, *537* (7620), 320–327. https://doi.org/10.1038/nature19946.

(2) Yeates, T. O.; Liu, Y.; Laniado, J. The Design of Symmetric Protein Nanomaterials Comes of Age in Theory and Practice. *Current Opinion in Structural Biology* 2016, *39*, 134–143. https://doi.org/10.1016/j.sbi.2016.07.003.

(3) Cannon, K. A.; Ochoa, J. M.; Yeates, T. O. High-Symmetry Protein Assemblies: Patterns and Emerging Applications. *Current Opinion in Structural Biology* 2019, *55*, 77–84. https://doi.org/10.1016/j.sbi.2019.03.008.

(4) Fletcher, J. M.; Harniman, R. L.; Barnes, F. R. H.; Boyle, A. L.; Collins, A.; Mantell, J.; Sharp, T. H.; Antognozzi, M.; Booth, P. J.; Linden, N.; Miles, M. J.; Sessions, R. B.; Verkade, P.; Woolfson, D. N. Self-Assembling Cages from Coiled-Coil Peptide Modules. *Science* 2013, *340* (6132), 595–599. https://doi.org/10.1126/science.1233936.

(5) Ljubetič, A.; Lapenta, F.; Gradišar, H.; Drobnak, I.; Aupič, J.; Strmšek, Ž.; Lainšček, D.; Hafner-Bratkovič, I.; Majerle, A.; Krivec, N.; Benčina, M.; Pisanski, T.; Veličković, T. Ć.; Round, A.; Carazo, J. M.; Melero, R.; Jerala, R. Design of Coiled-Coil Protein-Origami Cages That Self-Assemble *in Vitro* and *in Vivo*. *Nature Biotechnology* 2017, *35* (11), 1094–1101. https://doi.org/10.1038/nbt.3994.

(6) Sasaki, E.; Böhringer, D.; Van De Waterbeemd, M.; Leibundgut, M.; Zschoche, R.; Heck, A. J. R.; Ban, N.; Hilvert, D. Structure and Assembly of Scalable Porous Protein Cages. *Nature Communications* 2017, *8*, 1–10. https://doi.org/10.1038/ncomms14663.

(7) Suzuki, Y.; Cardone, G.; Restrepo, D.; Zavattieri, P. D.; Baker, T. S.; Tezcan, F. A. Self-Assembly of Coherently Dynamic, Auxetic, Two-Dimensional Protein Crystals. *Nature* 2016, *533* (7603), 369–373. https://doi.org/10.1038/nature17633.

(8) Gonen, S.; DiMaio, F.; Gonen, T.; Baker, D. Design of Ordered Two-Dimensional Arrays Mediated by Noncovalent Protein-Protein Interfaces. *Science* 2015, *348* (6241), 1365–1368. https://doi.org/10.1126/science.aaa9897.

(9) Lanci, C. J.; MacDermaid, C. M.; Kang, S.; Acharya, R.; North, B.; Yang, X.; Qiu, X. J.; DeGrado, W. F.; Saven, J. G. Computational Design of a Protein Crystal. *PNAS* 2012, *109* (19), 7304–7309. https://doi.org/10.1073/pnas.1112595109.

(10) Sciore, A.; Su, M.; Koldewey, P.; Eschweiler, J. D.; Diffley, K. A.; Linhares, B. M.; Ruotolo, B. T.; Bardwell, J. C. A.; Skiniotis, G.; Marsh, E. N. G. Flexible, Symmetry-Directed Approach to Assembling Protein Cages. *PNAS* 2016, *113* (31), 8681–8686. https://doi.org/10.1073/pnas.1606013113.

(11) Badieyan, S.; Sciore, A.; Eschweiler, J. D.; Koldewey, P.; Cristie-David, A. S.; Ruotolo, B. T.; Bardwell, J. C. A.; Su, M.; Marsh, E. N. G. Symmetry-Directed Self-Assembly of a Tetrahedral Protein Cage Mediated by de Novo-Designed Coiled Coils. *ChemBioChem* 2017, *18* (19), 1888–1892. https://doi.org/10.1002/cbic.201700406.

(12) Cristie-David, A. S.; Chen, J.; Nowak, D. B.; Bondy, A. L.; Sun, K.; Park, S. I.; Banaszak Holl, M. M.; Su, M.; Marsh, E. N. G. Coiled-Coil-Mediated Assembly of an Icosahedral Protein Cage with Extremely High Thermal and Chemical Stability. *J. Am. Chem. Soc.* 2019, *141* (23), 9207–9216. https://doi.org/10.1021/jacs.8b13604.

(13) Malay, A. D.; Miyazaki, N.; Biela, A.; Chakraborti, S.; Majsterkiewicz, K.; Stupka, I.; Kaplan, C. S.; Kowalczyk, A.; Piette, B. M. A. G.; Hochberg, G. K. A.; Wu, D.; Wrobel, T. P.; Fineberg, A.; Kushwah, M. S.; Kelemen, M.; Vavpetič, P.; Pelicon, P.; Kukura, P.; Benesch, J. L. P.; Iwasaki, K.; Heddle, J. G. An Ultra-Stable Gold-Coordinated Protein Cage Displaying Reversible Assembly. *Nature* 2019, *569* (7756), 438–442. https://doi.org/10.1038/s41586-019-1185-4.

(14) Lach, M.; Künzle, M.; Beck, T. Proteins as Sustainable Building Blocks for the Next Generation of Bioinorganic Nanomaterials. *Biochemistry* 2019, *58* (3), 140–141. https://doi.org/10.1021/acs.biochem.8b00966.

(15) Padilla, J. E.; Colovos, C.; Yeates, T. O. Nanohedra: Using Symmetry to Design Self Assembling Protein Cages, Layers, Crystals, and Filaments. *Proceedings of the National Academy of Sciences of the United States of America* 2001, *98* (5), 2217–2221. https://doi.org/10.1073/pnas.041614998.

(16) Yeates, T. O. Geometric Principles for Designing Highly Symmetric Self-Assembling Protein Nanomaterials. *Annual Review of Biophysics* 2017, *46* (1), 23–42. https://doi.org/10.1146/annurev-biophys-070816-033928.

(17) King, N. P.; Sheffler, W.; Sawaya, M. R.; Vollmar, B. S.; Sumida, J. P.; Andre, I.; Gonen, T.; Yeates, T. O.; Baker, D. Computational Design of Self-Assembling Protein Nanomaterials with Atomic Level Accuracy. *Science* 2012, *336* (6085), 1171–1174. https://doi.org/10.1126/science.1219364.

(18) Ringler, P.; Schulz, G. E. Self-Assembly of Proteins into Designed Networks. *Science* 2003, *302* (5642), 106–109. https://doi.org/10.1126/science.1088074.

(19) Sinclair, J. C.; Davies, K. M.; Vénien-Bryan, C.; Noble, M. E. M. Generation of Protein Lattices by Fusing Proteins with Matching Rotational Symmetry. *Nature Nanotechnology* 2011, *6* (9), 558–562. https://doi.org/10.1038/nnano.2011.122.

(20) Lai, Y. T.; Cascio, D.; Yeates, T. O. Structure of a 16-Nm Cage Designed by Using Protein Oligomers. *Science* 2012, *336* (6085), 1129–1129. https://doi.org/10.1126/science.1219351.

(21) Kwon, N.-Y.; Kim, Y.; Lee, J.-O. The Application of Helix Fusion Methods in Structural Biology. *Curr. Opin. Struct. Biol.* 2020, *60*, 110–116. https://doi.org/10.1016/j.sbi.2019.12.007.

(22) Lai, Y.-T.; Jiang, L.; Chen, W.; Yeates, T. O. On the Predictability of the Orientation of Protein Domains Joined by a Spanning Alpha-Helical Linker. *Protein Eng Des Sel* 2015, *28* (11), 491–500. https://doi.org/10.1093/protein/gzv035.

(23) Lai, Y.-T.; Reading, E.; Hura, G. L.; Tsai, K.-L.; Laganowsky, A.; Asturias, F. J.; Tainer, J. a.; Robinson, C. V.; Yeates, T. O. Structure of a Designed Protein Cage That Self-Assembles into a Highly Porous Cube. *Nature Chemistry* 2014, *6* (12), 1065–1071. https://doi.org/10.1038/nchem.2107.

(24) Cannon, K. A.; Nguyen, V. N.; Morgan, C.; Yeates, T. O. Design and Characterization of an Icosahedral Protein Cage Formed by a Double-Fusion Protein Containing Three Distinct Symmetry Elements. *ACS Synth. Biol.* 2020, *9* (3), 517–524. https://doi.org/10.1021/acssynbio.9b00392.

(25) King, N. P.; Bale, J. B.; Sheffler, W.; McNamara, D. E.; Gonen, S.; Gonen, T.; Yeates, T. O.; Baker, D. Accurate Design of Co-Assembling Multi-Component Protein Nanomaterials. *Nature* 2014, *510* (7503), 103–108. https://doi.org/10.1038/nature13404.

(26) Bale, J. B.; Gonen, S.; Liu, Y.; Sheffler, W.; Ellis, D.; Thomas, C.; Cascio, D.; Yeates, T. O.; Gonen, T.; King, N. P.; Baker, D. Accurate Design of Megadalton-Scale Two-Component Icosahedral Protein Complexes. *Science* 2016, *353* (6297), 389–395. https://doi.org/10.5061/dryad.8c65s.

(27) Hsia, Y.; Bale, J. B.; Gonen, S.; Shi, D.; Sheffler, W.; Fong, K. K.; Nattermann, U.; Xu, C.; Huang, P.-S.; Ravichandran, R.; Yi, S.; Davis, T. N.; Gonen, T.; King, N. P.; Baker, D. Design of a Hyperstable 60-Subunit Protein Icosahedron. *Nature* 2016, *535* (7610), 136–139. https://doi.org/10.1038/nature18010.

(28) Ueda, G.; Antanasijevic, A.; Fallas, J. A.; Sheffler, W.; Copps, J.; Ellis, D.; Hutchinson, G. B.; Moyer, A.; Yasmeen, A.; Tsybovsky, Y.; Park, Y.-J.; Bick, M. J.; Sankaran, B.; Gillespie, R. A.; Brouwer, P. J.; Zwart, P. H.; Veesler, D.; Kanekiyo, M.; Graham, B. S.; Sanders, R. W.; Moore, J. P.; Klasse, P. J.; Ward, A. B.; King, N. P.; Baker, D. Tailored Design of Protein Nanoparticle Scaffolds for Multivalent Presentation of Viral Glycoprotein Antigens. *eLife* 2020, *9*, e57659. https://doi.org/10.7554/eLife.57659.

(29) Bale, J. B.; Park, R. U.; Liu, Y.; Gonen, S.; Gonen, T.; Cascio, D.; King, N. P.; Yeates, T. O.; Baker, D. Structure of a Designed Tetrahedral Protein Assembly Variant Engineered to Have Improved Soluble Expression. *Protein Science* 2015, *24* (10), 1695–1701. https://doi.org/10.1002/pro.2748.

(30) Boyken, S. E.; Chen, Z.; Groves, B.; Langan, R. A.; Oberdorfer, G.; Ford, A.; Gilmore, J. M.; Xu, C.; DiMaio, F.; Pereira, J. H.; Sankaran, B.; Seelig, G.; Zwart, P. H.; Baker, D. De Novo Design of Protein Homo-Oligomers with Modular Hydrogen-Bond Network–Mediated Specificity. *Science* 2016, *352* (6286), 680–687. https://doi.org/10.1126/science.aad8865.

(31) Cannon, K. A.; Park, R. U.; Boyken, S. E.; Nattermann, U.; Yi, S.; Baker, D.; King, N. P.; Yeates, T. O. Design and Structure of Two New Protein Cages Illustrate Successes and Ongoing Challenges in Protein Engineering. *Protein Science* 2020, *29* (4), 919–929. https://doi.org/10.1002/pro.3802.

(32) Miyamoto, T.; Hayashi, Y.; Yoshida, K.; Watanabe, H.; Uchihashi, T.; Yonezawa, K.; Shimizu, N.; Kamikubo, H.; Hirota, S. Construction of a Quadrangular Tetramer and a Cage-Like Hexamer from Three-Helix Bundle-Linked Fusion Proteins. *ACS Synth. Biol.* 2019, *8* (5), 1112–1120. https://doi.org/10.1021/acssynbio.9b00019.

(33) McConnell, S. A.; Cannon, K. A.; Morgan, C.; McAllister, R.; Amer, B. R.; Clubb, R. T.; Yeates, T. O. Designed Protein Cages as Scaffolds for Building Multienzyme Materials. *ACS Synth. Biol.* 2020, *9* (2), 381–391. https://doi.org/10.1021/acssynbio.9b00407.

(34) Phippen, S. W.; Stevens, C. A.; Vance, T. D. R.; King, N. P.; Baker, D.; Davies, P. L. Multivalent Display of Antifreeze Proteins by Fusion to Self-Assembling Protein Cages Enhances Ice-Binding Activities. *Biochemistry* 2016, *55* (49), 6811–6820. https://doi.org/10.1021/acs.biochem.6b00864.

(35) Marcandalli, J.; Fiala, B.; Ols, S.; Perotti, M.; de van der Schueren, W.; Snijder, J.; Hodge, E.; Benhaim, M.; Ravichandran, R.; Carter, L.; Sheffler, W.; Brunner, L.; Lawrenz, M.; Dubois, P.; Lanzavecchia, A.; Sallusto, F.; Lee, K. K.; Veesler, D.; Correnti, C. E.; Stewart, L. J.; Baker, D.; Loré, K.; Perez, L.; King, N. P. Induction of Potent Neutralizing Antibody Responses by a Designed Protein Nanoparticle Vaccine for Respiratory Syncytial Virus. *Cell* 2019, *176* (6), 1420-1431.e17. https://doi.org/10.1016/j.cell.2019.01.046.

(36) Butterfield, G. L.; Lajoie, M. J.; Gustafson, H. H.; Sellers, D. L.; Nattermann, U.; Ellis, D.; Bale, J. B.; Ke, S.; Lenz, G. H.; Yehdego, A.; Ravichandran, R.; Pun, S. H.; King, N. P.; Baker, D. Evolution of a Designed Protein Assembly Encapsulating Its Own RNA Genome. *Nature* 2017, *552* (7685), 415–420. https://doi.org/10.1038/nature25157.

(37) Edwardson, T. G. W.; Mori, T.; Hilvert, D. Rational Engineering of a Designed Protein Cage for SiRNA Delivery. *Journal of the American Chemical Society* 2018, *140* (33), 10439–10442. https://doi.org/10.1021/jacs.8b06442.

(38) Liu, Y.; Gonen, S.; Gonen, T.; Yeates, T. O. Near-Atomic Cryo-EM Imaging of a Small Protein Displayed on a Designed Scaffolding System. *PNAS* 2018, *115* (13), 3362–3367. https://doi.org/10.1073/pnas.1718825115.

(39) Liu, Y.; Huynh, D. T.; Yeates, T. O. A 3.8 Å Resolution Cryo-EM Structure of a Small Protein Bound to an Imaging Scaffold. *Nature Communications* 2019, *10* (1), 1864. https://doi.org/10.1038/s41467-019-09836-0.

(40) Fletcher, J. M.; Boyle, A. L.; Bruning, M.; Bartlett, G. J.; Vincent, T. L.; Zaccai, N. R.; Armstrong, C. T.; Bromley, E. H. C.; Booth, P. J.; Brady, R. L.; Thomson, A. R.; Woolfson, D. N. A Basis Set of de Novo Coiled-Coil Peptide Oligomers for Rational Protein Design and Synthetic Biology. *ACS Synth. Biol.* 2012, *1* (6), 240–250. https://doi.org/10.1021/sb300028q.

(41) O'Shea, E. K.; Lumb, K. J.; Kim, P. S. Peptide 'Velcro': Design of a Heterodimeric Coiled Coil. *Current Biology* 1993, *3* (10), 658–667. https://doi.org/10.1016/0960-9822(93)90063-T.

(42) Gradišar, H.; Božič, S.; Doles, T.; Vengust, D.; Hafner-Bratkovič, I.; Mertelj, A.; Webb, B.; Šali, A.; Klavžar, S.; Jerala, R. Design of a Single-Chain Polypeptide Tetrahedron Assembled from Coiled-Coil Segments. *Nature Chemical Biology* 2013, *9* (6), 362–366. https://doi.org/10.1038/nchembio.1248.

(43) Reinke, A. W.; Grant, R. A.; Keating, A. E. A Synthetic Coiled-Coil Interactome Provides Heterospecific Modules for Molecular Engineering. *J. Am. Chem. Soc.* 2010, *132* (17), 6025–6031. https://doi.org/10.1021/ja907617a.

(44) Ogihara, N. L.; Weiss, M. S.; Eisenberg, D.; Degrado, W. F. The Crystal Structure of the Designed Trimeric Coiled Coil Coil-VaLd: Implications for Engineering Crystals and Supramolecular Assemblies. *Protein Science* 1997, *6* (1), 80–88. https://doi.org/10.1002/pro.5560060109.

(45) Nautiyal, S.; Woolfson, D. N.; King, D. S.; Alber, T. A Designed Heterotrimeric Coiled Coil. *Biochemistry* 1995, *34* (37), 11645–11651. https://doi.org/10.1021/bi00037a001.

(46) Glover, J. N. M.; Harrison, S. C. Crystal Structure of the Heterodimeric BZIP Transcription Factor C-Fos–c-Jun Bound to DNA. *Nature* 1995, *373* (6511), 257–261. https://doi.org/10.1038/373257a0.

(47) O'Shea, E. K.; Rutkowski, R.; Stafford, W. F.; Kim, P. S. Preferential Heterodimer Formation by Isolated Leucine Zippers from Fos and Jun. *Science* 1989, *245* (4918), 646–648. https://doi.org/10.1126/science.2503872.

(48) Punjani, A.; Rubinstein, J. L.; Fleet, D. J.; Brubaker, M. A. CryoSPARC: Algorithms for Rapid Unsupervised Cryo-EM Structure Determination. *Nature Methods* 2017, *14* (3), 290–296. https://doi.org/10.1038/nmeth.4169.

(49) Heinig, M.; Frishman, D. STRIDE: A Web Server for Secondary Structure Assignment from Known Atomic Coordinates of Proteins. *Nucleic Acids Res* 2004, *32* (suppl_2), W500–W502. https://doi.org/10.1093/nar/gkh429.

(50) Studier, F. W. Protein Production by Auto-Induction in High-Density Shaking Cultures. *Protein Expression and Purification* 2005, *41* (1), 207–234. https://doi.org/10.1016/j.pep.2005.01.016.

(51) Kabsch, W. XDS. *Acta Cryst D* 2010, *66* (2), 125–132. https://doi.org/10.1107/S0907444909047337.

(52) McCoy, A. J.; Grosse-Kunstleve, R. W.; Adams, P. D.; Winn, M. D.; Storoni, L. C.; Read, R. J. Phaser Crystallographic Software. *J Appl Cryst* 2007, *40* (4), 658–674. https://doi.org/10.1107/S0021889807021206.

(53) Emsley, P.; Cowtan, K. Coot: Model-Building Tools for Molecular Graphics. *Acta Cryst D* 2004, *60* (12), 2126–2132. https://doi.org/10.1107/S0907444904019158.

(54) Adams, P. D.; Grosse-Kunstleve, R. W.; Hung, L.-W.; Ioerger, T. R.; McCoy, A. J.; Moriarty, N. W.; Read, R. J.; Sacchettini, J. C.; Sauter, N. K.; Terwilliger, T. C. PHENIX: Building New Software for Automated Crystallographic Structure Determination. *Acta Cryst D* 2002, *58* (11), 1948–1954. https://doi.org/10.1107/S0907444902016657.

(55) Scheres, S. H. W. RELION: Implementation of a Bayesian Approach to Cryo-EM Structure Determination. *Journal of Structural Biology* 2012, *180* (3), 519–530. https://doi.org/10.1016/j.jsb.2012.09.006.

(56) Glover, J. N. M.; Harrison, S. C. Crystal Structure of the Heterodimeric BZIP Transcription Factor C-Fos–c-Jun Bound to DNA. *Nature* 1995, *373* (6511), 257–261. https://doi.org/10.1038/373257a0.

(57) O'Shea, E. K.; Rutkowski, R.; Stafford, W. F.; Kim, P. S. Preferential Heterodimer Formation by Isolated Leucine Zippers from Fos and Jun. *Science* 1989, *245* (4918), 646–648. https://doi.org/10.1126/science.2503872.

(58) O'Shea, E. K.; Lumb, K. J.; Kim, P. S. Peptide 'Velcro': Design of a Heterodimeric Coiled Coil. *Current Biology* 1993, *3* (10), 658–667. https://doi.org/10.1016/0960-9822(93)90063-T.

# CHAPTER 3

# A complete rule set for designing symmetry combination materials from protein molecules

# A complete rule set for designing symmetry combination materials from protein molecules

**Joshua Laniado**[a] **and Todd O. Yeates**[a,b,c,1]

[a]Molecular Biology Institute, University of California, Los Angeles, CA 90095; [b]Department of Energy Institute for Genomics and Proteomics, University of California, Los Angeles, CA 90095; and [c]Department of Chemistry and Biochemistry, University of California, Los Angeles, CA 90095

Edited by Edward H. Egelman, University of Virginia, Charlottesville, VA, and approved October 21, 2020 (received for review July 22, 2020)

**Diverse efforts in protein engineering are beginning to produce novel kinds of symmetric self-assembling architectures, from protein cages to extended two-dimensional (2D) and three-dimensional (3D) crystalline arrays. Partial theoretical frameworks for creating symmetric protein materials have been introduced, but no complete system has been articulated. Only a minute fraction of the possible design space has been explored experimentally, in part because that space has not yet been described in theory. Here, in the form of a multiplication table, we lay out a complete rule set for materials that can be created by combining two chiral oligomeric components (e.g., proteins) in precise configurations. A unified system is described for parameterizing and searching the construction space for all such symmetry-combination materials (SCMs). In total, 124 distinct types of SCMs are identified, and then proven by computational construction. Mathematical properties, such as minimal ring or circuit size, are established for each case, enabling strategic predictions about potentially favorable design targets. The study lays out the theoretical landscape and detailed computational prescriptions for a rapidly growing area of protein-based nanotechnology, with numerous underlying connections to mathematical networks and chemical materials such as metal organic frameworks.**

protein design | symmetry | nanotechnology | self-assembly | biomaterials

A central goal of bottom-up nanotechnology is to create materials or supramolecular architectures by self-assembly with atomic precision (1). Experimental studies in the last two decades have demonstrated considerable progress toward that goal using diverse kinds of molecules as building blocks, including organic and metal complexes (2–5), nucleic acids (6–8), and peptides (9–12). Recent developments have brought an increasing focus on using oligomeric protein molecules as a basis for broad programs in materials design. Several types of self-assembling materials have been demonstrated using protein molecules as building blocks (11, 13–31), with principles of symmetry providing an important foundation (32). The power of symmetry-based synthesis was recognized by early work in supramolecular coordination chemistry (33). Related ideas emerged in the area of protein-based design in 2001 with a strategy for combining simple component symmetries in the form of small protein oligomers to create complex supramolecular architectures, including self-assembling cubic cages and extended materials with repeating symmetries (13). For proteins, Padilla et al. (13) provided specific design rules for several types of symmetric materials, but the possibilities articulated were limited in their scope, primarily because at that time the database of known protein structures was not so richly populated with diverse oligomeric components. The current abundance of known protein building blocks with diverse symmetries (34–38), and recent successful applications of symmetry-based design ideas, now motivate a deeper question. What is the full scope of the geometric forms that can be achieved by combining simpler oligomeric building blocks in precisely defined ways?

Systemization plays a key role in scientific exploration—evolutionary phylogeny, star systems, subatomic particles, and the periodic table are just a few examples—with tables, and mathematical trees or networks often providing the organizing framework. In the area of protein structure, tertiary (39, 40) and quaternary structural forms have been systematized in various contexts (41–43). The field of symmetry-based materials design lends itself naturally to organization as a multiplication table, with each cell indicating the outcome from combining two simpler component symmetries to create a more complex architecture. In the present work, we provide a complete multiplication table that articulates design outcomes, along with construction rules to cover the field of protein materials created by combinations of two simpler symmetry types. We introduce the name SCMs (symmetry combination materials), to describe the broad space of materials that are possible. This theoretical and computational study, which effectively completes the foundation begun 20 y ago by Padilla et al. (13), provides a blueprint and an enabling framework for a blossoming area of macromolecular design.

## Results

**Group Principles for Combined Symmetries.** When two (or more) separate symmetry types are brought together in some geometrically defined way, a higher symmetry is generated. What results is the mathematical group generated by multiplying the symmetry operations of the separate symmetries together, repeatedly if necessary. Stated another way, the result is the smallest mathematical group that contains the two component symmetries as subgroups. If the two components are two different symmetric protein oligomers, and a geometrically specific connection, covalent or noncovalent, can be introduced between them, then a defined protein-based SCM can be created. It follows from the closure properties of a group that, if the protein subunits of the first oligomer type are connected together in some defined orientation (e.g., by noncovalent interfaces), and likewise for the second oligomer type, and if a rigid connection

## Significance

Building nanoscale materials in a predictable way is a major goal in broad fields of molecular engineering. In pursuing that goal, protein molecules are gaining interest as a source of structurally defined, self-assembling building blocks, but methods and geometric rules for how they might be combined into novel materials have not been fully laid out yet. This work brings together ideas from structural biology, geometry, and group theory to articulate a complete rule set that describes all the kinds of protein-based materials that can be created by combining separate oligomeric components, thereby laying the foundation for an emerging area of science and biotechnology.

Author contributions: J.L. and T.O.Y. designed research, performed research, analyzed data, and wrote the paper.

The authors declare no competing interest.

This article is a PNAS Direct Submission.

[1]To whom correspondence may be addressed. Email: yeates@mbi.ucla.edu.

This article contains supporting information online at https://www.pnas.org/lookup/suppl/doi:10.1073/pnas.2015183117/-/DCSupplemental.

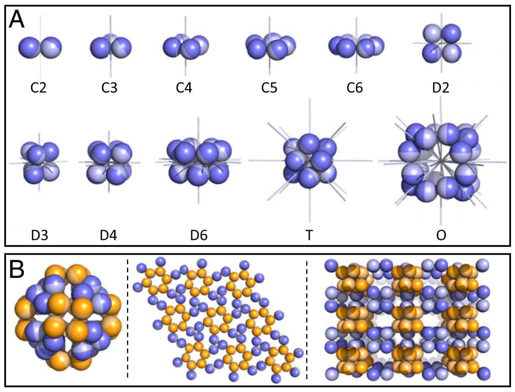


**Fig. 1.** Diagrams of symmetric oligomeric building blocks and example two-component SCMs. (*A*) Illustration of point group symmetries (top row) C2, C3, C4, C5, C6, D2, (bottom row) D3, D4, D6, T, and O with their symmetry axes. (*B*) A finite assembly with octahedral symmetry constructed by combining a C3 trimer (blue) and a C4 tetramer (orange)—O:{C3}{C4} (*Left*). An extended p6 2D layer formed by combining a C3 trimer (blue) and a C6 hexamer (orange)—p6:{C3}{C6} (*Middle*). A P422 3D crystalline array assembled by combining a D2 tetramer (blue) and a D4 octamer (orange)—P422:{D2}{D4} (*Right*).

can be established between the two oligomeric components, then the entire configuration—however large it may be, even extending indefinitely—will be connected together in an architecture of defined symmetry and structure. These principles were laid out by Padilla et al. (13); there and in subsequent work, diverse strategies for bringing protein oligomers together in defined ways have resulted in several types of novel protein materials (11, 14, 15, 17–21). Three geometric examples are shown in Fig. 1, one resulting in a cubic cage (a finite assembly), another in an extended 2D layer and the other in a three-dimensional (3D) crystal.

Combining simpler symmetries to create more highly symmetric products is an idea rooted in mathematics. There are many cases, however, where this underlying mathematical idea leads to hypothetical assemblies that are physically impossible, or nearly impossible, for components with compact/globular shapes, owing to unavoidable steric collisions. Such cases need to be identified to avoid including recipes for designing many theoretical constructions that could not be realized in the context of materials built from typical protein molecules. Two illustrative cases where mathematically allowed combinations of symmetries lead to physically impossible/improbable assemblies are shown in *SI Appendix*, Fig. S1. In the simplest example, a twofold axis of symmetry is combined with a threefold axis of symmetry that is parallel and coincident (overlapping) in space. The mathematical result is a sixfold axis of symmetry. In other words, C2 symmetry combined or multiplied by C3 symmetry (with coincident axes) gives C6 symmetry. However, attempting to connect a dimeric protein to a trimeric protein with their symmetry axes overlapping leads to collision rather than a hexameric ($a_6b_6$) structure; three copies of the dimer would fall on top of each other, as would two copies of the trimer. Problematic cases such as this, of which there are a great variety, can only be rescued by elaborate entwinement of elongated (noncompact) shapes (*SI Appendix*, Fig. S1). These complex situations had to be analyzed and systematically removed so that the design space described would ultimately consist only of plausible materials. A more complete description of the problem of recognizing entwinement cases is described in *Methods* and *SI Appendix*, Text.

We note certain caveats to the completeness of the system of symmetry combinations to be articulated here. First, we require the symmetries of the two oligomers to be of the ordinary point symmetry types (i.e., with subunits related to each other by pure rotational operations). Such cases dominate among natural protein oligomers. Therefore, for our purposes, the possibility of an individual component with a built-in helical symmetry is ignored. The second exclusion relates to the difference between assemblies built from one type of component vs. two different components. Symmetry principles apply as well to building materials from a single oligomeric component (14), by considering that creating a new mode of self-association between different copies of a single oligomer type effectively introduces a new symmetry operation. Such designs could involve built-in screw axis operations. We also note that a much greater design set is possible by considering combinations of more than two symmetric components; one case of a three-component design has been published recently (30). Finally, further discussions here focus on high order point symmetries and extended materials in two and three dimensions (i.e., molecular layers and crystals), but not on filamentous assemblies that extend in only one direction. Extensions of the symmetry ideas to those systems is discussed in *SI Appendix*, Text. Accordingly, we direct our attention to the more complex design problems in two and three dimensions.

**Articulating All Symmetric Outcomes.** The set of all possible symmetry combinations is expansive, but the analysis can be simplified by first examining underlying point group symmetries, i.e., temporarily setting aside translational repetition that might be present in the resulting (combined) symmetry. The number of different point groups possible for biological molecules (i.e., lacking mirror operations) is small enough that the outcomes of all pairwise combinations can be enumerated with relative ease. A systematic approach is possible with the following rule: If symmetry groups A and B (considered in some defined orientation) are both subgroups of symmetry C, and if no lower subgroup of C also contains A and B (orientated as before) as subgroups, then symmetries A and B will combine to generate symmetry C. As an example, octahedral symmetry O contains C2 and C3 symmetries at an intersecting angle of 35.3°, and no lower symmetry group does so. Therefore, C2 and C3 intersecting at 35.3° will generate point group O. Further analysis and a complete table are provided in *Methods* and in *SI Appendix*, Table S1.

In laying out all possible point group symmetry combinations (above), the symmetry axes are taken to be intersecting (at the center point of the finite structure). However, many more outcomes are possible when the component symmetries are combined with their axes or origins offset. Such cases cannot generate finite cage structures, and instead give rise to extended materials based on infinite symmetry groups, i.e., layer groups and space groups. Additional levels of complexity are faced in systematically determining all of the allowable ways in which the 17 possible layer groups or 65 possible space groups for proteins might be realized. First, as described above, knowing the underlying point symmetry that results from any given combination of component symmetries limits the possibilities that need to be examined. Second, the dimensionality (i.e., a 2D layer or 3D crystal) of the resulting symmetry can be determined by considering the nature of any nonintersecting symmetry axes and whether the resulting symmetry is isotropic (*SI Appendix*, Text and Fig. S7). Finally, once the dimensionality is established for a candidate symmetry combination, often the correct result must be selected from multiple space groups having the same underlying point group symmetry. Common cases are those of cubic space groups that differ from each other only by screw axis designations or by lattice centering. The correct outcomes in those cases can be discerned by examining the standard crystallographic space group tables, noting that

**Table 1. A complete multiplication table for SCM materials created by two (chiral) oligomeric components**

| | C2 | C3 | | C4 | | C5 | C6 | D2 | | D3 | | | D4 | | D6 | T | | O | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C2 | 2,2<br>**D2**<br>4 2 | 2,3<br>**p6**<br>4 6 | 2,3<br>**D3**<br>4 2 | 2,4<br>**p4**<br>4 4 | 2,4<br>**D4**<br>4 2 | 2,5<br>**D5**<br>4 2 | 2,6<br>**p6**<br>4 3 | 2,4<br>**c222**<br>4 4 | 2,4<br>**p422**<br>3 4 | 2,6<br>**p312**<br>3 3 | 2,6<br>**R32**<br>4 4 | 2,6<br>**p622$_a$**<br>3 2 | 2,8<br>**p422$_a$**<br>3 2 | 2,8<br>**p422$_b$**<br>3 2 | 2,12<br>**p622$_a$**<br>3 2 | 2,12<br>**P23**<br>3 2 | | 2,24<br>**P432$_a$**<br>3 2 | 2,24<br>**F432$_a$**<br>3 2 |
| | | 2,3<br>**p312**<br>5 6 | 2,3<br>**T**<br>4 3 | 2,4<br>**p42$_1$2**<br>5 4 | 2,4<br>**O**<br>4 3 | | 2,6<br>**D6**<br>4 2 | 2,4<br>**I4$_1$22**<br>4 6 | 2,4<br>**p622**<br>3 3 | 2,6<br>**p622$_b$**<br>3 2 | 2,6<br>**P6$_3$22**<br>4 4 | 2,6<br>**F4$_1$32**<br>3 3 | 2,8<br>**I422**<br>4 4 | 2,8<br>**P432**<br>3 3 | 2,12<br>**p622$_b$**<br>3 2 | 2,12<br>**F23**<br>3 3 | | 2,24<br>**F432$_b$**<br>3 2 | 2,24<br>**P432$_b$**<br>3 2 |
| | | 2,3<br>**I2$_1$3**<br>5 10 | 2,3<br>**O**<br>4 4 | 2,4<br>**I432**<br>5 8 | | 2,5<br>**I**<br>4 3 | 2,6<br>**p622**<br>5 4 | 2,4<br>**P6$_2$22**<br>4 6 | 2,4<br>**I432**<br>3 4 | 2,6<br>**I4$_1$32$_a$**<br>3 2 | 2,6<br>**I432**<br>3 4 | 2,6<br>**I4$_1$32$_b$**<br>3 2 | 2,8<br>**I432**<br>3 2 | | 2,12<br>**P622**<br>4 4 | 2,12<br>**F4$_1$32**<br>3 2 | | 2,24<br>**I432**<br>3 2 | |
| | | 2,3<br>**P4$_1$32**<br>5 10 | 2,3<br>**I**<br>4 5 | | | | | 2,4<br>**I4$_1$32**<br>3 3 | | 2,6<br>**P4$_1$32**<br>3 3 | | | | | | | | | |
| C3 | | 3,3<br>**p3**<br>4 3 | 3,3<br>**T**<br>4 2 | 3,4<br>**O**<br>4 2 | 3,4<br>**F432**<br>5 6 | 3,5<br>**I**<br>4 2 | 3,6<br>**p6**<br>4 2 | 3,4<br>**p622**<br>3 2 | 3,4<br>**P23**<br>3 3 | 3,6<br>**p312**<br>3 2 | 3,6<br>**p321**<br>3 2 | 3,6<br>**P4$_1$32**<br>3 4 | 3,8<br>**P432**<br>3 2 | | 3,12<br>**p622**<br>3 2 | 3,12<br>**F23**<br>3 2 | | 3,24<br>**F432**<br>3 2 | |
| | | 3,3<br>**P2$_1$3**<br>5 5 | | | | | | 3,4<br>**F432**<br>3 3 | 3,4<br>**I4$_1$32**<br>3 2 | | | | | | | | | | |
| C4 | | | | 4,4<br>**p4**<br>4 2 | 4,4<br>**P432**<br>5 4 | | | 4,4<br>**p422**<br>3 2 | 4,4<br>**p42$_1$2**<br>3 2 | 4,6<br>**I432**<br>3 2 | | | 4,8<br>**p422**<br>3 2 | 4,8<br>**P432**<br>3 2 | | 4,12<br>**F432**<br>3 2 | | 4,24<br>**P432**<br>3 2 | |
| | | | | | | | | 4,4<br>**I432**<br>3 2 | 4,4<br>**F432**<br>3 3 | | | | | | | | | | |
| C6 | | | | | | | | 6,4<br>**p622**<br>3 2 | | 6,6<br>**p622**<br>2 2 | | | | | | | | | |
| D2 | | | | | | | | 4,4<br>**p222**<br>2 2 | 4,4<br>**F222**<br>3 3 | *2,3*<br>**p622**<br>1 1 | 4,6<br>**P622**<br>2 2 | | *2,4*<br>**p422**<br>1 1 | 4,8<br>**P422**<br>2 2 | *2,6*<br>**p622**<br>1 1 | *2,6*<br>**P23**<br>1 1 | 4,12<br>**P23**<br>1 2 | *2,12*<br>**F432**<br>1 1 | |
| | | | | | | | | 4,4<br>**P4$_2$22**<br>2 2 | 4,4<br>**P6$_2$22**<br>2 2 | 4,6<br>**P4$_2$32**<br>1 2 | *2,3*<br>**I4$_1$32**<br>1 1 | | 4,8<br>**I422**<br>2 2 | *2,4*<br>**I432**<br>1 1 | 4,12<br>**P622**<br>2 2 | *2,6*<br>**F432**<br>1 1 | 4,12<br>**P4$_2$32**<br>1 2 | 4,24<br>**I432**<br>1 2 | |
| | | | | | | | | 4,4<br>**P4$_2$32**<br>1 2 | | | | | | | | | | | |
| D3 | | | | | | | | | | *3,3*<br>**p312**<br>1 1 | 6,6<br>**P312**<br>2 2 | 6,6<br>**P6$_3$22**<br>2 2 | 6,8<br>**I432**<br>1 2 | | *3,6*<br>**p622**<br>1 1 | *2,4*<br>**F4$_1$32**<br>1 1 | | *2,8*<br>**I432**<br>1 1 | |
| | | | | | | | | | | 6,6<br>**P4$_2$32**<br>1 2 | *3,3*<br>**P4$_1$32**<br>1 1 | | | | 6,12<br>**P622**<br>2 2 | | | | |
| D4 | | | | | | | | | | | | | *4,4*<br>**p422**<br>1 1 | 8,8<br>**P422**<br>2 2 | | | | *2,6*<br>**P432**<br>1 1 | |
| | | | | | | | | | | | | | *4,4*<br>**P432**<br>1 1 | | | | | *4,12*<br>**P432**<br>1 1 | |
| T | | | | | | | | | | | | | | | | *4,4*<br>**F23**<br>1 1 | *6,6*<br>**F23**<br>1 1 | *4,8*<br>**F432**<br>1 1 | |
| O | | | | | | | | | | | | | | | | | | *8,8*<br>**P432**<br>1 1 | *6,6*<br>**F432**<br>1 1 |

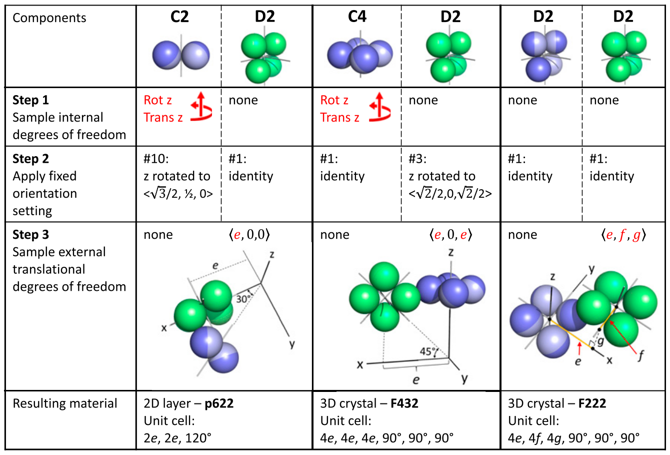
Finite assemblies are highlighted in green, 2D layer groups in pink, and 3D space groups in blue. For each SCM, the resulting space group (bold), coordination numbers for the two components (top), degrees of freedom for design (bottom left), and ring size (bottom right) are provided. Coordination numbers affected by symmetry about a shared axis between oligomers are italicized. For entries with chiral space groups, the enantiomeric space group is allowed but not listed here to be concise. Detailed parameters for construction are provided in *SI Appendix*, Table S3 and Dataset S1.

the correct symmetry must be one that has so-called "Wyckoff positions" matching the symmetries of the separate components.

A complete set of rules for combining two separate point group symmetries was prepared following the systematic analysis described above. The results can be cast as a multiplication table between two separate point group symmetries that would be contributed by two oligomeric protein components being brought together as building blocks (Table 1). The table of all 124 possibilities includes several design cases leading to finite cage-like assemblies; we have enumerated these before. A total of 35 different design cases lead to 2D SCMs. Of these, 6 are polar (meaning the two sides of the layer are distinct), while the other 29 are bipolar (with no distinction possible between top and bottom faces). Of the 35 layer possibilities, 33 are isotropic in the plane, implying equivalent physical properties in multiple distinct directions. The table describes 76 SCMs leading to 3D crystals. Six of these fall in chiral space groups. In total, 58 are cubic, meaning they are isotropic in all three spatial directions. There are six cases where a pair of distinct SCMs with the same resulting symmetry can be created from the same types of component symmetries in different arrangements (requiring a and b designations). Some of the design rules are straightforward, but many are nonobvious. Setting aside cases that could only be built from complex entangled molecular shapes (which we have eliminated), the list is, as far as we can establish from systematic study, complete and exhaustive. We undertook computational tests for mathematical correctness and physical constructability and proved the validity of the full set of SCMs (*SI Appendix*, Figs. S2–S4). The specific rules for their construction have not been described until now.

The large number of possible SCMs, together with the diverse molecular engineering methods being employed, motivate a general system for notation. We expand on the notation system used by King et al. (17), following principles used for describing the connectivity of chemical structures by SMILE strings. Here, individual oligomeric components (typically bearing their own internal symmetry) take the place of "atoms," with curly brackets used to enclose point symmetry designations. Different letter symbols describe the kinds of connections established between the components, effectively playing the role of bond types in SMILE strings: e.g., "H" for a continuous helical linker, "I" for novel noncovalent interface, "S" for disulfide bond, "F" for flexible fusion, "M" for metal site, with other possibilities allowed. The resulting symmetry is denoted first. As examples using this system: the original tetrahedrally (T) symmetric designed cage by Padilla et al. (13) would be T:{C3}H{C2}; the first two-component octahedron obtained by King et al., using interface design, would be O:{C3}I{C2}; a flexibly connected octahedral cage obtained by Sciore et al. (25) would be O:{C4}F{C3}, and so on. This scheme is precise but flexible enough to describe diverse types of SCMs.

**Procedural Construction for Rigid Body Sampling.** The rules for designing various constructions naturally take the form of restrictions—constraints or equations that must be satisfied, e.g., by the angles between symmetry axes, the distances between them when they do not intersect, and so on. However, in order to enable a general system for construction, the problem must be cast in reverse, essentially as a problem in characterizing the dual space. King et al. provided a description of that approach for the relatively straightforward case of constructing point groups (for creating protein cages) (14, 17) using the framework of symmetry definition files in Rosetta (14, 15, 17–19, 44). A unified system is called for to cover the complete set of 2D and 3D SCMs defined here. Depending on the SCM, one or more of the oligomeric components may carry internal degrees of rigid body rotation and translation (e.g., of a cyclic oligomer about its symmetry axis) (Fig. 2) (*SI Appendix*, Fig. S6). Further external degrees of freedom apply to relative shifts between the two components. The values of these shifts ultimately relate to unit cell parameters in the resulting materials, sometimes in nonobvious ways. Note that between application of internal and external degrees of freedom, certain fixed rotational settings are applied in each case. For example, the oligomeric components are understood to begin in canonical orientations with symmetry axes along principle direction, whereas a specific rotation matrix might be needed to rotate a threefold axis of symmetry to fall along the body diagonal of a resulting cubically symmetric material. For every type of symmetry



| Components | C2 | D2 | C4 | D2 | D2 | D2 |
|---|---|---|---|---|---|---|
| **Step 1** Sample internal degrees of freedom | Rot z Trans z | none | Rot z Trans z | none | none | none |
| **Step 2** Apply fixed orientation setting | #10: z rotated to <√3/2, ½, 0> | #1: identity | #1: identity | #3: z rotated to <√2/2,0,√2/2> | #1: identity | #1: identity |
| **Step 3** Sample external translational degrees of freedom | none | $\langle e, 0, 0\rangle$ | none | $\langle e, 0, e\rangle$ | none | $\langle e, f, g\rangle$ |
| Resulting material | 2D layer – **p622** Unit cell: 2*e*, 2*e*, 120° | | 3D crystal – **F432** Unit cell: 4*e*, 4*e*, 4*e*, 90°, 90°, 90° | | 3D crystal – **F222** Unit cell: 4*e*, 4*f*, 4*g*, 90°, 90°, 90° | |

**Fig. 2.** Procedural construction for rigid body sampling, diagrammed for three example SCMs. The examples illustrate how the rules can be implemented in a stepwise procedure that enables the allowed construction space to be sampled. In each case, the rigid body sampling degrees of freedom are highlighted in red. The examples are all cases where the number of degrees of freedom is 3, although they present in different forms. The orientation setting matrices are given in *SI Appendix*, Table S1. *SI Appendix*, Table S3 provides the full listing of degrees of freedom for sampling every construction type (available as Dataset S1). Further construction details are provided for the middle example (F432:{C4}{D2}) in *SI Appendix*, Fig. S6. Construction protocols for all 124 SCMs are described in pseudocode in *SI Appendix*, Text (also available as Dataset S2).

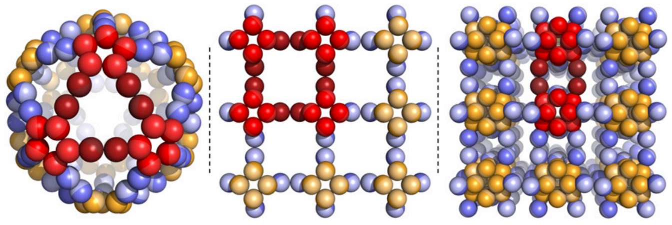


**Fig. 3.** Illustration of the concept of ring size for three example SCMs. The ring size is the number of oligomers of each type in a closed circuit. I:{C2}{C5}, a finite assembly with icosahedral symmetry constructed by combining a C2 dimer (blue) and a C5 pentamer (orange), has a ring size of 3 (*Left*). p4:{C2}{C4}, a 2D layer with p4 symmetry formed by combining a C2 dimer (blue) and a C4 tetramer (orange), has a ring size of 4 (*Middle*). P23:{C2}{T}, a 3D crystalline array with P23 symmetry assembled by combining a C2 dimer (blue) and a T dodecamer (orange), has a ring size of 2 (*Right*). For each case, a single ring is illustrated with participating oligomers in two shades of red.

combination, we have articulated the available rigid body degrees of freedom and a set of parameters (*SI Appendix*, Table S3) describing a stepwise procedure (*SI Appendix*, Fig. S6) for computationally sampling the available space of configurations. Our encoding therefore provides the necessary framework for sampling fully the symmetry combination design space, enabling various program applications. To emphasize how readily the system is reduced to practice, we produce pseudocode (which embodies all of the required symmetry-based information) for performing design searches for all 124 SCMs (*SI Appendix*, Text and Dataset S2).

**Geometric Characteristics of Symmetric Design Types.** Materials created by different symmetry combinations exhibit distinct geometric and structural properties (described in Table 1). Some of these properties describe important material characteristics. Material dimensionality (e.g., cage, layer, or crystal), isotropy (uniformity in multiple directions), and layer polarity were noted earlier. The total degrees of freedom available for design are also an important determinant. This value, ranging from 1 to 5 for different cases, relates to designability. SCMs with multiple degrees of freedom for construction offer deeper search spaces for bringing the separate components together, giving much greater chances for finding low energy configurations. Materials created by connecting multiple components have network-like properties that can be understood via mathematical graphs: i.e., nodes and edges. The connectivity or "coordination number" in these nets, is dictated by the order of the oligomeric symmetry (divided by the rotational symmetry at an edge in some cases). Another key property of networks is the size of the rings, i.e., the minimum number of edges in a closed circuit (Fig. 3). Previous experimental studies on designed protein assemblies have suggested that designs where the ring size is large may be practically difficult to achieve, since accidental collapse of partially formed rings might occur whenever there is sufficient flexibility during the assembly process (21). We have calculated the ring size mathematically for all SCMs (*Methods*). The possible values vary by case, remarkably, from 1 to 10 (Table 1). The special cases of ring size equal to 1 occur when two adjacent oligomers A and B are connected (e.g., by fusion or interface design) through more than one of their subunits (22). Conversely, several SCMs have surprisingly large ring sizes, and could represent particularly challenging design targets (*SI Appendix*, Fig. S5).

## Discussion

The systematization of symmetry combinations provides a blueprint for a vast space for designed molecular materials. This blueprint—along with thousands of known protein-based building blocks and emerging methods to connect them together in precisely defined ways—provides a uniquely predictable system for building nanomaterials based on symmetry principles. Our unified system for describing and parameterizing the space of molecular constructions enables immediate applications within diverse computational platforms.

The completeness of the system furthermore provides a rational basis for experimental prioritization. Based on their unique mathematical and geometric characteristics, various SCMs offer distinct advantages and disadvantages. One can prospect for SCMs that might hold multiple characteristics generally expected to be favorable. The number of degrees of freedom for design (high being favorable) and the ring size (low anticipated to be favorable) are two examples. Being able to predict favorable assembly types for 3D crystals could be particularly important. Symmetric protein or polypeptide oligomers have been designed to crystallize in a few cases (11, 23), but almost all of the space for fully predictive design of 3D crystals remains unexplored. Based on our analysis, among the possible crystal SCMs, a few interesting cases stand out as being potentially privileged choices. A total of 19 3D SCMs have at least three degrees of freedom for construction and a ring size of no greater than 2. Among these, one might focus on cases where the coordination numbers are greater than 2 to favor more connected networks, but where the components are lower than cubic symmetry (T and O) since those are not so common among natural structures. Those criteria single out five SCMs, all giving cubic type crystals, which could be favorable targets for novel crystal designs. Under our proposed nomenclature, these would be I4(1)32:{C3}x{D2}; I432:{C4}x{D2}; P432:{C4}x{D4}; P432:{C3}x{D4}; and I432:{C4}x{D3}. The latter two offer the further advantage that the pair of component symmetries do not allow any other alternative symmetric outcomes, as Table 1 shows; those two SCMs are diagrammed in Fig. 4. Other criteria, weighed differently, could of course suggest other favorable SCM targets.

The work here offers important connections to mathematics, materials chemistry, and efforts to design ordered nanomaterials from other kinds of molecules (4, 45, 46). Connections to chemistry can be drawn by seeing the precisely defined orientations conferred by protein-protein interactions as an extension of "directional bonding" ideas from early work in supramolecular coordination chemistry (47). In terms of unifying mathematical ideas, foundations for describing regular nets in three dimensions were laid out in a classic text by Wells in 1977 (48). Delgado-Friedrichs, O'Keeffe, Yaghi, and other workers (4, 5, 49–57) began expanding on those ideas nearly 20 y ago to describe the rich diversity of symmetric materials formed by metal organic frameworks (MOFs). Some aspects of MOF nets translate well to our protein-based SCMs, whereas other aspects lead to points of departure (as described in *SI Appendix*, Text). Our full articulation of two-component SCMs should advance the subject of protein-based materials design in the same way that systematization of rules and components has advanced the field of MOFs (58). Finally, the generality of the building scheme laid out here

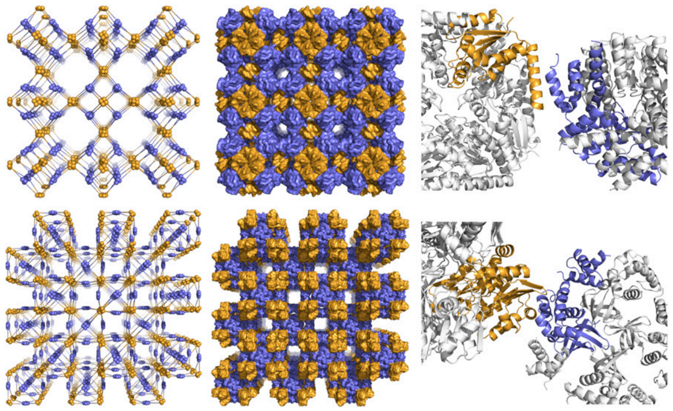


**Fig. 4.** Potentially privileged SCMs for 3D crystal designs. A P432 crystal constructed by combining a C3 trimer (blue) and a D4 octamer (orange) (*Top*). A crystal with space group I432 constructed by combining a C4 tetramer (blue) and a D3 hexamer (orange) (*Bottom*). The SCMs illustrated here have three degrees of freedom available for construction and a ring size of 2. Furthermore, in both cases, the combination of the two symmetry types can only give rise to a single type of 3D space group. The SCMs are illustrated as networks on the *Left*. A plausible protein packing is modeled for each case (*Middle*). A single redesigned contact between the hypothetically docked oligomers is necessary and sufficient to hold the respective modeled architectures together (*Right*). The protein constructions are shown to convey design plausibility. Additional candidates are possible with different choices for the component oligomers; multiple plausible docking modes are sometimes possible with the same oligomers in different orientations. The examples shown are therefore only representatives of a multitude of plausible candidates and were selected based on criteria suitable for interface design (*SI Appendix*, Text). Other criteria are possible for alternate interfacial modes and connection types (Protein Data Bank ID codes: *Top*, 2Q0T, blue, and 4B4K, orange; *Bottom*, 1CUK, blue, and a D3 hexamer, 2V78, orange).

could transcend the molecular scale, with diverse applications to engineering novel materials across much larger length scales (1).

## Methods

**Enumeration of Point Group Symmetry Combinations.** The symmetry combination rule is that if symmetry groups A and B (considered in some defined orientation) are both subgroups of symmetry C, and if no lower subgroup of C also contains A and B (orientated as before) as subgroups, then symmetries A and B will combine to generate symmetry C. The text explains the example of combining C2 with C3 at an angle of 35.3° to generate octahedral symmetry, O. There is no symmetry lower than O that contains C2 and C3 at an angle of 35.3°. A further example is helpful, now considering what group is generated by C2 and C3 at an intersecting angle of 54.7°. Symmetry O contains those elements intersecting at 54.7° (by way of the C2 axis contained as a subgroup of the C4 axis in O), yet symmetry O is not generated. Instead, symmetry T (tetrahedral) is generated, as it also contains C2 and C3 axes intersecting at 54.7°, and it is lower symmetry than (i.e., a subgroup of) O. A complete analysis, including orientation specifications, is provided in *SI Appendix*, Table S1. Dihedral symmetries Dn with $n = 5$ or $n > 6$, which could contribute to filament or rod structures but not to cubic point symmetries or extended materials in two and three dimensions, are noted in *SI Appendix*, Text. Quasiperiodic packings, e.g., Penrose tilings, are not considered.

**Determination of Dimensionality (2D Layer or 3D Crystal).** If there is no offset between the two components (i.e., their symmetry axes all intersect at a single point), then the result will be a finite point group symmetry. An example case was diagrammed in Fig. 1 comprising a protein cage or cluster with cubic symmetry. If any symmetry axes do not intersect, then an extended material must result. If this is the case, and the underlying point group created by the combination of component symmetries is cubic (T or O), then the result will be a 3D crystal; this follows from the isotropy of cubic symmetries. If, however, the resulting underlying point group symmetry is only cyclic or dihedral, then a 3D or 2D material will arise, depending on more complex rules, e.g., depending on whether or not the combined symmetries present any twofold axes of symmetry (perpendicular to the unique polar axis) that are nonintersecting. For completeness, a decision flowchart describing how the dimensionality of the resulting material depends on the arrangement of component symmetries is provided in *SI Appendix*, Fig. S7.

**Analysis of Practically Implausible Symmetry Combinations.** Several problematic categories were identified where mathematically legal symmetry group combinations are forbidden for compact shapes. The first is described in the text; it occurs when the combination of symmetries leads to a product group wherein the Wykoff symmetry where one (or both) of the component oligomers sits is higher than the symmetry of the oligomer itself. In this category, the example case of C6:{C2}{C3} is illustrated in *SI Appendix*, Fig. S1*A*. A second problematic category where entanglement is unavoidable arises when the two different components fall on Wykoff positions that are related to each other by symmetry (*SI Appendix*, Fig. S1*B*). A third category arises when oligomer 1 cannot reach the necessary instance of oligomer 2 without colliding with other unintended molecular components. These situations, which had to be systematically eliminated, are described in more detail in *SI Appendix*, Text.

**Constructability Calculations.** Motivated by the complexity of the process required to lay out a complete set of design rules, we undertook computational tests for mathematical correctness and physical constructability. For mathematical correctness, for every SCM entry in the rules table, we constructed the two sets of matrix and translation operations corresponding to the component symmetries, suitably oriented and translated according to the specified rules, and confirmed by multiplication and expansion that symmetry elements of the expected type were produced. For tests of constructability, we assessed whether each combination could be plausibly realized by packings of compact molecular components. Model oligomers based on simple spherical subunits were used as building blocks. In each construction case, the parameters from the rules table were applied as a search (i.e., in a nested loop over free parameters). See the section on pseudocode in *SI Appendix*, Text, for how such a search is implemented. Noncolliding configurations where the two oligomeric components were in contact were identified. We found that a few of our prospective SCMs were more difficult to construct than others. Among a list of 125 entries that had been collated manually according to the symmetry rules in the text and above, only one (I432:{C4}{D2} noted above) was ultimately ruled unconstructible owing to unavoidable collisions. In one case, F432:{C3}{C4}, packing without collisions using spherical subunits was problematic, but model oligomers based on elongated subunits (i.e., a subunit composed of two spheres instead of one) could be packed successfully; this case was retained. Models illustrating packing of compact shapes for all 124 SCMs were rendered in PyMol for 3D visualization (available at https://people.mbi.ucla.edu/yeates/SCM_files/). These calculations validated all 124 SCMs (*SI Appendix*, Figs. S2–S4) along with the detailed parameterizations provided in the SCM table (Table 1 and *SI Appendix*, Table S3 and Dataset S1).

**Mathematical Evaluation of Ring Size.** The value we assign for ring size describes the number of each type of oligomer traversed in the smallest ring: the ring size would be 3 for a cycle described by $–A_i–B_x–A_j–B_y–A_k–B_z–$, where A and B are the two oligomer types and the subscripts indicate different copies of each oligomer. The ring size, *r*, is evaluated mathematically as follows:

*r* is the minimum value of *n* for which

$$(A_{i1}B_{j1})(A_{i2}B_{j2})\ldots(A_{in}B_{jn}) = \text{Identity},$$

for some choice of operations $A_{i1},\ldots,A_{in}$ and $B_{j1},\ldots,B_{jn}$ (none of them equal to the identity operation) taken from the two symmetry groups A and B (suitably oriented and positioned). We evaluated the ring size for every SCM in Table 1 computationally by exhaustive matrix multiplication.

**Data Availability.** All study data are included in the article and *SI Appendix*.

**ACKNOWLEDGMENTS.** This work was supported by NSF Grant CHE-1629214. We thank Kyle Meador, Kevin Cannon, members of the T.O.Y. laboratory, Duilio Cascio, Michael Sawaya, and Alex Lisker for discussions and computing support. We thank Neil King, Frank DiMaio, Will Sheffler, Jacob Bale, and David Baker for discussions during early stages of this work. We also thank Omar Yaghi, Akif Tezcan, and Sarah Teichmann for helpful comments on the manuscript.

1. G. M. Whitesides, B. Grzybowski, Self-assembly at all scales. *Science* **295**, 2418–2421 (2002).
2. M. Fujita, Metal-directed self-assembly of two- and three-dimensional synthetic receptors. *Chem. Soc. Rev.* **27**, 417 (1998).
3. D. Fujita *et al*., Self-assembly of tetravalent Goldberg polyhedra from 144 small components. *Nature* **540**, 563–566 (2016).
4. M. O'Keeffe, M. A. Peskov, S. J. Ramsden, O. M. Yaghi, The reticular chemistry structure resource (RCSR) database of, and symbols for, crystal nets. *Acc. Chem. Res.* **41**, 1782–1789 (2008).
5. H. Li, M. Eddaoudi, M. O'Keeffe, O. M. Yaghi, Design and synthesis of an exceptionally stable and highly porous metal-organic framework. *Nature* **402**, 276–279 (1999).
6. J. Zheng *et al*., From molecular to macroscopic via the rational design of a self-assembled 3D DNA crystal. *Nature* **461**, 74–77 (2009).
7. E. Winfree, F. Liu, L. A. Wenzler, N. C. Seeman, Design and self-assembly of two-dimensional DNA crystals. *Nature* **394**, 539–544 (1998).
8. M. R. Jones, N. C. Seeman, C. A. Mirkin, Nanomaterials. Programmable materials and the nature of the DNA bond. *Science* **347**, 1260901 (2015).
9. J. L. Beesley, D. N. Woolfson, The de novo design of α-helical peptides for supramolecular self-assembly. *Curr. Opin. Biotechnol.* **58**, 175–182 (2019).
10. L. Regan *et al*., Protein design: Past, present, and future. *Biopolymers* **104**, 334–350 (2015).
11. C. J. Lanci *et al*., Computational design of a protein crystal. *Proc. Natl. Acad. Sci. U.S.A.* **109**, 7304–7309 (2012).
12. G. Grigoryan *et al*., Computational design of virus-like protein assemblies on carbon nanotube surfaces. *Science* **332**, 1071–1076 (2011).
13. J. E. Padilla, C. Colovos, T. O. Yeates, Nanohedra: Using symmetry to design self assembling protein cages, layers, crystals, and filaments. *Proc. Natl. Acad. Sci. U.S.A.* **98**, 2217–2221 (2001).
14. N. P. King *et al*., Computational design of self-assembling protein nanomaterials with atomic level accuracy. *Science* **336**, 1171–1174 (2012).
15. J. B. Bale *et al*., Accurate design of megadalton-scale two-component icosahedral protein complexes. *Science* **353**, 389–394 (2016).
16. A. D. Malay *et al*., An ultra-stable gold-coordinated protein cage displaying reversible assembly. *Nature* **569**, 438–442 (2019).
17. N. P. King *et al*., Accurate design of co-assembling multi-component protein nanomaterials. *Nature* **510**, 103–108 (2014).
18. Y. Hsia *et al*., Corrigendum: Design of a hyperstable 60-subunit protein icosahedron. *Nature* **540**, 150 (2016).
19. S. Gonen, F. DiMaio, T. Gonen, D. Baker, Design of ordered two-dimensional arrays mediated by noncovalent protein-protein interfaces. *Science* **348**, 1365–1368 (2015).
20. Y.-T. Lai, D. Cascio, T. O. Yeates, Structure of a 16-nm cage designed by using protein oligomers. *Science* **336**, 1129 (2012).
21. Y.-T. Lai *et al*., Structure of a designed protein cage that self-assembles into a highly porous cube. *Nat. Chem.* **6**, 1065–1071 (2014).
22. J. C. Sinclair, K. M. Davies, C. Vénien-Bryan, M. E. M. Noble, Generation of protein lattices by fusing proteins with matching rotational symmetry. *Nat. Nanotechnol.* **6**, 558–562 (2011).
23. P. A. Sontz, J. B. Bailey, S. Ahn, F. A. Tezcan, A metal organic framework with spherical protein nodes: Rational chemical design of 3D protein crystals. *J. Am. Chem. Soc.* **137**, 11598–11601 (2015).
24. J. M. Fletcher *et al*., Self-assembling cages from coiled-coil peptide modules. *Science* **340**, 595–599 (2013).
25. A. Sciore *et al*., Flexible, symmetry-directed approach to assembling protein cages. *Proc. Natl. Acad. Sci. U.S.A.* **113**, 8681–8686 (2016).
26. P. Ringler, G. E. Schulz, Self-assembly of proteins into designed networks. *Science* **302**, 106–109 (2003).
27. Y. Suzuki *et al*., Self-assembly of coherently dynamic, auxetic, two-dimensional protein crystals. *Nature* **533**, 369–373 (2016).
28. J. F. Matthaei *et al*., Designing two-dimensional protein arrays through fusion of multimers and interface mutations. *Nano Lett.* **15**, 5235–5239 (2015).
29. E. Golub *et al*., Constructing protein polyhedra via orthogonal chemical interactions. *Nature* **578**, 172–176 (2020).
30. K. A. Cannon, V. N. Nguyen, C. Morgan, T. O. Yeates, Design and characterization of an icosahedral protein cage formed by a double-fusion protein containing three distinct symmetry elements. *ACS Synth. Biol.* **9**, 517–524 (2020).
31. Y. Azuma, T. G. W. Edwardson, D. Hilvert, Tailoring lumazine synthase assemblies for bionanotechnology. *Chem. Soc. Rev.* **47**, 3543–3557 (2018).
32. T. O. Yeates, Geometric principles for designing highly symmetric self-assembling protein nanomaterials. *Annu. Rev. Biophys.* **46**, 23–42 (2017).
33. D. L. Caulder, K. N. Raymond, The rational design of high symmetry coordination clusters. *J. Chem. Soc. Dalton Trans.* **1999**, 1185–1200 (1999).
34. E. Krissinel, K. Henrick, Inference of macromolecular assemblies from crystalline state. *J. Mol. Biol.* **372**, 774–797 (2007).
35. K. A. Cannon, J. M. Ochoa, T. O. Yeates, High-symmetry protein assemblies: Patterns and emerging applications. *Curr. Opin. Struct. Biol.* **55**, 77–84 (2019).
36. S. Dey, D. W. Ritchie, E. D. Levy, PDB-wide identification of biological assemblies from conserved quaternary structure geometry. *Nat. Methods* **15**, 67–72 (2018).
37. E. D. Levy, E. Boeri Erba, C. V. Robinson, S. A. Teichmann, Assembly reflects evolution of protein complexes. *Nature* **453**, 1262–1265 (2008).
38. S. Bliven, A. Lafita, A. Parker, G. Capitani, J. M. Duarte, Automated evaluation of quaternary structures from protein crystals. *PLoS Comput. Biol.* **14**, e1006104 (2018).
39. A. G. Murzin, S. E. Brenner, T. Hubbard, C. Chothia, SCOP: A structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.* **247**, 536–540 (1995).
40. C. A. Orengo *et al*., CATH–A hierarchic classification of protein domain structures. *Structure* **5**, 1093–1108 (1997).
41. S. E. Ahnert, J. A. Marsh, H. Hernández, C. V. Robinson, S. A. Teichmann, Principles of assembly reveal a periodic table of protein complexes. *Science* **350**, aaa2245 (2015).
42. D. L. Caspar, A. Klug, Physical principles in the construction of regular viruses. *Cold Spring Harb. Symp. Quant. Biol.* **27**, 1–24 (1962).
43. E. D. Levy, J. B. Pereira-Leal, C. Chothia, S. A. Teichmann, 3D complex: A structural classification of protein complexes. *PLoS Comput. Biol.* **2**, e155 (2006).
44. F. DiMaio, A. Leaver-Fay, P. Bradley, D. Baker, I. André, Modeling symmetric macromolecular structures in Rosetta3. *PLoS One* **6**, e20450 (2011).
45. C. R. Simmons *et al*., Construction and structure determination of a three-dimensional DNA crystal. *J. Am. Chem. Soc.* **138**, 10047–10054 (2016).
46. N. C. Seeman, Nucleic acid junctions and lattices. *J. Theor. Biol.* **99**, 237–247 (1982).
47. B. J. Holliday, C. A. Mirkin, Strategies for the construction of supramolecular compounds through coordination chemistry. *Angew. Chem. Int. Ed. Engl.* **40**, 2022–2043 (2001).
48. A. F. Wells, *Three-Dimensional Nets and Polyhedra* (Wiley, 1977).
49. O. M. Yaghi *et al*., Reticular synthesis and the design of new materials. *Nature* **423**, 705–714 (2003).
50. O. M. Yaghi, G. Li, H. Li, Selective binding and removal of guests in a microporous metal–organic framework. *Nature* **378**, 703–706 (1995).
51. C. Bonneau, O. Delgado-Friedrichs, M. O'Keeffe, O. M. Yaghi, Three-periodic nets and tilings: Minimal nets. *Acta Crystallogr. A* **60**, 517–520 (2004).
52. O. Delgado Friedrichs, M. O'Keeffe, O. M. Yaghi, Three-periodic nets and tilings: Regular and quasiregular nets. *Acta Crystallogr. A* **59**, 22–27 (2003).
53. O. Delgado Friedrichs, M. O'Keeffe, O. M. Yaghi, Three-periodic nets and tilings: Semiregular nets. *Acta Crystallogr. A* **59**, 515–525 (2003).
54. O. Delgado-Friedrichs, M. O'Keeffe, O. M. Yaghi, Taxonomy of periodic nets and the design of materials. *Phys. Chem. Chem. Phys.* **9**, 1035–1043 (2007).
55. O. Delgado-Friedrichs, M. O'Keeffe, O. M. Yaghi, Three-periodic nets and tilings: Edge-transitive binodal structures. *Acta Crystallogr. A* **62**, 350–355 (2006).
56. N. W. Ockwig, O. Delgado-Friedrichs, M. O'Keeffe, O. M. Yaghi, Reticular chemistry: Occurrence and taxonomy of nets and grammar for the design of frameworks. *Acc. Chem. Res.* **38**, 176–182 (2005).
57. O. Delgado-Friedrichs, M. O'Keeffe, Identification of and symmetry computation for crystal nets. *Acta Crystallogr. A* **59**, 351–360 (2003).
58. M. J. Kalmutzki, N. Hanikel, O. M. Yaghi, Secondary building units as the turning point in the development of the reticular chemistry of MOFs. *Sci. Adv.* **4**, eaat9180 (2018).

**Supplementary Information for:**

# A Complete Rule Set for Designing Symmetry Combination Materials from Protein Molecules

Joshua Laniado and Todd O. Yeates

Contact Information: Todd Yeates – yeates@mbi.ucla.edu

**This PDF file includes:**

Supplementary Text

Supplementary Tables S1 – S3

Supplementary Figures S1 – S7

SI References

**Other supplementary materials for this manuscript include the following:**

Datasets S1 to S2

**Bipolar Filament and Rod Group Symmetries**

The ideas of symmetry combination apply to filaments and rods (1–5). The simplest type of bipolar filament was described and demonstrated experimentally by Padilla et al. (1), with other types of bipolar filaments or rods demonstrated in later studies (2, 3). The symmetry rules can be described simply, while also allowing all possible orders *n* of dihedral symmetry Dn. As this does not lend itself to a tabular form – i.e. it would require rows and columns for component types useful for nothing other than rods – we have elected to describe the filament/rod group symmetries as separate constructions rather than as tabular entries in the main symmetry combination table. Two basic construction forms are possible:

*Type 1*
Components: Dn (any n) + C2
Construction geometry: 2-fold axis of C2 perpendicular and intersecting the n-fold axis of Dn, and not in the plane formed by the 2-fold axes of Dn.
Degrees of freedom: 4
Ring size: 2
Special case: with n=1 for Dn, the situation becomes two dimers with non-intersecting axes, as described by Padilla *et al.* (1). The degrees of freedom increase to 5, and the ring size is undefined.

*Type 2*
Components: Dn + Dn (any n)
Construction geometry: Coincident n-fold axes of symmetry for the two components (refs 2, 3).
Degrees of freedom: 2
Ring size: 1

Various polar filaments and rods, not based on point group combinations but on other principles, have also been described (4–8).

**Isotropy**

Here we say that a symmetry is isotropic if a second rank material tensor (e.g. stress or strain) that is invariant under the given symmetry must be isotropic in form (i.e. a scalar times the identity matrix) and therefore invariant under all rotations. Higher rank tensors (e.g the fourth rank stiffness tensor) could obey the specified symmetry without being isotropic.

**Generalization of the Symmetry Combination Notation System**

The notation system introduced here can be generalized to cover other systems. Single component designs, for example, could be treated as a case of zero-length ring closure as allowed by SMILE strings, using numerical subscripts. One of King's single component tetrahedra, obtained by introducing a homotypic 2-fold interface at the surface of a cyclic trimer, would be T:{C3}$_1$I$_{[C2]}$, where the subscript 1 (without recurrence) indicates a self-interaction at component 1, and the designed interface denoted by "I" carries a notation for the symmetry it introduces. Expansion to more than two components is likewise straightforward. The 3-component icosahedron described recently by Cannon *et al.* would be I:{C5}H{C2}F{C3}(ref 9). An example from the full table of symmetry outcomes described in this study (entry 62 in Table 1) would be the formation of a cubic 3-D crystal from a trimer and a dihedral tetramer, F432:{C3}x{D2}, with 'x' as a placeholder for whatever type of connection might be used to connect the trimer to the tetramer.

## **Rigid body degrees of freedom**

The rigid body degrees of freedom for any given SCM fall into different categories. Cyclic oligomers carry two 'internal' degrees of freedom, one rotational and one translational, along their unique symmetry axes. [Special rules apply: e.g. if both components are cyclic and their symmetry axes are parallel, then one of the translational degrees of freedom is redundant and must be sacrificed.] Higher symmetry component oligomers (dihedral and cubic) do not carry internal degrees of freedom. Next, depending on the specific design type, certain 'external' (translational) degrees of freedom are available to specify separation distances or vectors between symmetry axes or between symmetry centers of the components. Values for these external degrees of freedom are ultimately related to the unit cell parameters of the repeating SCMs that are generated. To unify the scheme, oligomers are taken to sit initially in canonical orientations and positions (e.g. with their symmetry axis along z). Internal degrees of freedom are applied in this starting frame of reference. Prior to applying external (translational) degrees of freedom, specific fixed rotational settings are applied where necessary to bring the axes of a canonically oriented oligomer into some required orientation.

Ultimately, for every design case, the number of degrees of freedom must be complementary to the number of underlying geometric restrictions. There are 6 total degrees of freedom to describe the relative relationship between two 3-dimensonal objects, such as two oligomeric protein building blocks, so the number of degrees of freedom for construction must be 6 minus the number of equations conveyed by the geometric restrictions. As an example, Table 1 describes the formation of layer symmetry p42$_1$2 from a dimer and a cyclic tetramer, p42$_1$2:{C2}{C4}. The only geometric restriction for this construction is that the 2-fold and 4-fold symmetry axes must be perpendicular, non-intersecting but without restriction on their separation distance. The requirement for a 90° angle between two axes comprises a single equation or restriction to be subtracted from 6, so working from the geometric restrictions we conclude that the available rigid body space has dimension 5. Now, working in reverse as a construction by sampling free rigid body parameters, above we noted that cyclic oligomers each carry one internal rotational and one internal translational degree of freedom, so this provides 4 internal parameters together for the two cyclic oligomers. Then there is a single external degree of freedom describing the translational separation between the non-intersecting axes; the unit cell spacing will be twice this separation distance. Again, the calculated number of parameters comes to 5. Similar ideas apply to every case.

In addition to the continuous rigid body degrees of freedom, a special type of orientational degeneracy must be considered. Point symmetry groups lower than octahedral can be reoriented in ways that do not change the positions of the underlying symmetry axes: a cyclic oligomer can be flipped over; a D2 tetramer can be reoriented 6 different ways by exchanging x, y, and z axes, as examples. These orientational degeneracies must be allowed in order to execute a complete sampling of the rigid body parameter space for any given problem. We give the mathematical treatment of such degeneracies in Table S2.

**Interpretation of SCM tables (Table S3) as procedural samplings for construction**

Each SCM entry (Table S3) provides a complete description of the sampling required to evaluate constructions for a given pair of oligomeric protein components conforming to the required symmetries. Prior to construction, all oligomers are presumed to sit in canonical orientations and positions, as follows. Unique symmetry axes (e.g. in the case of cyclic of dihedral symmetries) are oriented along z. For dihedral symmetries, 2-fold symmetry axes perpendicular to the unique axis are oriented in the x-y plane, with one such axis along x. Cubic symmetry T is oriented with its D2 subgroup symmetry axes along x, y, and z. Symmetry O is oriented with its 4-fold symmetry axes along x, y, and z. The oligomers are positioned with their points of central symmetry (i.e. their centers of mass) at the origin.

'Internal' degrees of freedom for construction sampling (if they exist) are specified by a rotation and a translation. The axis of rotation is z in all cases considered here, and a variable 'a' specifies the rotation value to be applied. We annotate that operation as r:<0,0,1,a>. The translation belonging to the internal degrees of freedom (also along z in all cases described) is specified by the variable quantity b. We annotate that operation as t:<0,0,b>. Following application of rotations and translations associated with internal degrees of freedom, the oligomer is then rotated by a setting matrix (specified in the table); in many cases this is simply the identity matrix. Internal degrees of rotational and translational freedom may be associated with both oligomers, or only one, or neither. In order to assign distinct variable names, internal degrees of freedom for the second oligomer (if they exist) are annotated as r:<0,0,1,c> and t:<0,0,d>. The variables and operations associated with the internal degrees of freedom must be applied combinatorially, i.e. in nested loops. The final degrees of freedom (if they exist) are 'external' translations associated with shifts between the oligomers (after having applied internal degrees of freedom and setting matrices). Any external degrees of translational freedom are applied in innermost loop(s). Depending on the number of spatial degrees of freedom, these are described by variables e, f, and g. Each such variable quantity may express its effect through shifts in one or both oligomers, along directions that may or may not be principle directions. For example, <e,0,e> would indicate that the shift is along the x-z diagonal. If a shift of <e,0,0> is associate with oligomer 1 and <0,e,0> is associated with oligomer 2, then both oligomers are shifted according to the chosen value of e. Though the relative shift between oligomers is the primary concern, in many cases applying separate shifts to both component oligomers is important in order to situate the final assembly according to standard settings for crystallographic plane and space groups. Thereafter, expanding extended assemblies is convenient using established tables of plane and space group operators. Relationships between the external translational variables and the unit cell values for two or three dimensional materials are given in Table S3.

The interpretation of parameters in terms of rotations and translations to be applied to the component oligomers is diagrammed in Figure 2 in the main text, and in further detail for one example in Supplementary Figure S6, as it might be executed in a computer program. [Pseudocode is included below for every SCM.]

Following construction of the appropriate nested loops, applying the rotations and translations specified by the free loop variables will place the two oligomers in a spatial configuration representing a mathematically legal symmetry arrangement for the desired SCM. The remaining steps are associated with determining the suitability of each candidate configuration for a physical design. The vast majority of configurations sampled will not represent suitable configurations for establishing a connection between the two component oligomers, e.g. either being too far away from each other or colliding. Tests for collision typically require application of the full symmetry of the construction. This is relatively straightforward, since the parameterizations have been carefully laid out so that the resulting symmetry of the SCM can be applied in a canonical reference frame, as noted above for the crystal space groups.

Detailed tests for the designability of any given configuration will depend on the type of protein (or other) design being considered. For example, for a design based on genetic fusion of the two oligomeric components, the respective protein chain termini need to be proximal; if the fusion is to involve a continuous alpha helix, then the termini would furthermore have to be oriented for a helical connection. If a novel interface is to be designed, then an interface with sufficient area and shape complementarity is required as a starting point for computational sequence design. Designs based on other types of connection, such as metal coordination or disulfide bonding, would require proximity of residues amenable to substitution with appropriate amino acid side chains.

## Pseudocode for all SCM constructions

To illustrate the enablement of SCM design calculations, we wrote a computer script that takes the SCM data table as input, and a user-defined table entry number, and writes out pseudo code for executing that design type. The orientational and translational sampling is fully specified through loops over free parameters and application of necessary setting orientations. Pseudocode for sampling the construction space for all 124 SCM types are provided here and in readable text form in Supplementary Dataset 2.

```
# D2:{C2}{C2}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.0,1.0][0.0,1.0,0.0][-
1.0,0.0,0.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
```

```
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# p6:{C2}{C3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - loop over value of shift parameter e
          - apply shift to oligomer 1: <e,0,0>
          - apply shift to oligomer 2: <e,0.577350*e,0>
          - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# D3:{C2}{C3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.0,1.0][0.0,1.0,0.0][-
1.0,0.0,0.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# p312:{C2}{C3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,0.0,1.0][0.0,-
1.0,0.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - loop over value of shift parameter e
          - apply shift to oligomer 1: <e,0,0>
          - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# T:{C2}{C3}
- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over z rotations for oligomer 2
```

```
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# I213:{C2}{C3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
        - loop over value of shift parameter e
          - apply shift to oligomer 1: <0,e,0>
          - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# O:{C2}{C3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# P4132:{C2}{C3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
        - loop over value of shift parameter e
          - apply shift to oligomer 1: <2*e,e,0>
          - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# I:{C2}{C3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
```

```
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,0.934172,0.356822][0.0,-0.356822,0.934172]
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# p4:{C2}{C4}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - loop over value of shift parameter e
          - apply shift to oligomer 1: <e,0,0>
          - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# D4:{C2}{C4}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.0,1.0][0.0,1.0,0.0][-
1.0,0.0,0.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# p4212:{C2}{C4}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.707107,0.707107][0.0,-
0.707107,0.707107][1.0,0.0,0.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - loop over value of shift parameter e
          - apply shift to oligomer 2: <e,0,0>
          - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# O:{C2}{C4}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
```

```
    - apply fixed rotation setting to oligomer 1:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# I432:{C2}{C4}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - loop over value of shift parameter e
          - apply shift to oligomer 1: <2*e,e,0>
          - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# D5:{C2}{C5}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.0,1.0][0.0,1.0,0.0][-
1.0,0.0,0.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# I:{C2}{C5}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[0.850651,0.0,0.525732][0.0,1.0,0.0][-0.525732,0.0,0.850651]
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# p6:{C2}{C6}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
```

```
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - loop over value of shift parameter e
          - apply shift to oligomer 1: <e,0,0>
          - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# D6:{C2}{C6}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.0,1.0][0.0,1.0,0.0][-
1.0,0.0,0.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# p622:{C2}{C6}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,0.0,1.0][0.0,-
1.0,0.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - loop over value of shift parameter e
          - apply shift to oligomer 1: <e,0,0>
          - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# c222:{C2}{D2}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - loop over value of shift parameter f
        - apply shift to oligomer 1: <e,f,0>
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# p422:{C2}{D2}
```

```
- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.707107,0.707107][0.0,-
0.707107,0.707107][1.0,0.0,0.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <e,0,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# I4122:{C2}{D2}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.0,1.0][0.0,1.0,0.0][-
1.0,0.0,0.0]
    - apply fixed rotation setting to oligomer 2: [0.707107,0.707107,0.0][-
0.707107,0.707107,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - loop over value of shift parameter f
        - apply shift to oligomer 1: <0,e,f>
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

#########################################

# p622:{C2}{D2}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.5,0.866025][0.0,-
0.866025,0.5][1.0,0.0,0.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <e,0,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# P6222:{C2}{D2}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.5,0.866025][0.0,-
0.866025,0.5][1.0,0.0,0.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - loop over value of shift parameter f
        - apply shift to oligomer 1: <0,0,e>
        - apply shift to oligomer 2: <f,0,0>
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

#########################################

# I432:{C2}{D2}
```

```
- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - apply fixed rotation setting to oligomer 2: [0.707107,0.707107,0.0][-
0.707107,0.707107,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <2*e,0,e>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# I4132:{C2}{D2}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - apply fixed rotation setting to oligomer 2: [0.707107,0.707107,0.0][-
0.707107,0.707107,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <-2*e,3*e,0>
      - apply shift to oligomer 2: <0,2*e,e>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# p312:{C2}{D3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,0.0,1.0][0.0,-
1.0,0.0]
    - apply fixed rotation setting to oligomer 2: [0.0,-
1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,0,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# R32:{C2}{D3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.0,1.0][0.0,1.0,0.0][-
1.0,0.0,0.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - loop over value of shift parameter f
        - apply shift to oligomer 1: <0,e,f>
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# p622:{C2}{D3} – type a

- loop over z rotations for oligomer 1
```

```
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2: [0.0,-
1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,0,0>
      - apply shift to oligomer 2: <e,0.57735*e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# p622:{C2}{D3} – type b

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.0,1.0][0.0,1.0,0.0][-
1.0,0.0,0.0]
    - apply fixed rotation setting to oligomer 2: [0.0,-
1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <e,0.57735*e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# P6322:{C2}{D3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.0,1.0][0.0,1.0,0.0][-
1.0,0.0,0.0]
    - apply fixed rotation setting to oligomer 2: [0.0,-
1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - loop over value of shift parameter f
        - apply shift to oligomer 2: <e,0.57735*e,f>
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

#########################################

# F4132:{C2}{D3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <e,e,e>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# I4132:{C2}{D3} – type a

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
```

13

```
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <0,2*e,0>
      - apply shift to oligomer 2: <e,e,e>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# I432:{C2}{D3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - apply fixed rotation setting to oligomer 2: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <e,e,e>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# I4132:{C2}{D3} – type b

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - apply fixed rotation setting to oligomer 2: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <0,e,-2*e>
      - apply shift to oligomer 2: <e,e,e>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# P4132:{C2}{D3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - apply fixed rotation setting to oligomer 2: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <0,e,-2*e>
      - apply shift to oligomer 2: <3*e,3*e,3*e>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# p422:{C2}{D4} – type a

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
```

14

```
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,0,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# p422:{C2}{D4} – type b

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.0,1.0][0.0,1.0,0.0][-
1.0,0.0,0.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <0,e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# I422:{C2}{D4}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.707107,0.707107][0.0,-
0.707107,0.707107][1.0,0.0,0.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - loop over value of shift parameter f
        - apply shift to oligomer 1: <0,e,f>
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

#########################################

# P432:{C2}{D4}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <0,0,e>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# I432:{C2}{D4}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
```

```
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <2*e,e,0>
      - apply shift to oligomer 2: <2*e,2*e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# p622:{C2}{D6} – type a

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,0,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# p622:{C2}{D6} – type b

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,0.0,1.0][0.0,-
1.0,0.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,0,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# P622:{C2}{D6}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,0.0,1.0][0.0,-
1.0,0.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - loop over value of shift parameter f
        - apply shift to oligomer 1: <e,0,f>
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# P23:{C2}{T}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
```

```
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,0,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# F23:{C2}{T}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# F4132:{C2}{T}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <2*e,3*e,0>
      - apply shift to oligomer 2: <0,4*e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# P432:{C2}{O} – type a

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,0,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# F432:{C2}{O} – type a

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
```

```
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# F432:{C2}{O} – type b

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,0,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# P432:{C2}{O} – type b

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <0,e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# I432:{C2}{O}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <-e,e,e>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# p3:{C3}{C3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - loop over value of shift parameter e
```

```
          - apply shift to oligomer 2: <e,0.57735*e,0>
          - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# T:{C3}{C3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2: [0.707107,-
0.408248,0.577350][0.707107,0.408248,-0.577350][0.0,0.816497,0.577350]
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# P213:{C3}{C3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2: [0.707107,-
0.408248,0.577350][0.707107,0.408248,-0.577350][0.0,0.816497,0.577350]
        - loop over value of shift parameter e
          - apply shift to oligomer 2: <e,0,0>
          - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# O:{C3}{C4}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# F432:{C3}{C4}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
```

```
        - loop over value of shift parameter e
          - apply shift to oligomer 2: <e,0,0>
          - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# I:{C3}{C5}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,0.934172,0.356822][0.0,-0.356822,0.934172]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[0.850651,0.0,0.525732][0.0,1.0,0.0][-0.525732,0.0,0.850651]
        - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# p6:{C3}{C6}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - loop over value of shift parameter e
          - apply shift to oligomer 1: <e,0.57735*e,0>
          - test coordinates of the oligomers for a designable contact interaction &
write out candidates

##########################################

# p622:{C3}{D2}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,0.57735*e,0>
      - apply shift to oligomer 2: <e,0,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# P23:{C3}{D2}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
```

```
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <e,0,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# F432:{C3}{D2}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - apply fixed rotation setting to oligomer 2:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <e,0,e>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# I4132:{C3}{D2}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - apply fixed rotation setting to oligomer 2:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <2*e,e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# p312:{C3}{D3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2: [0.0,-
1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,0.57735*e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# p321:{C3}{D3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,0.57735*e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# P4132:{C3}{D3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.707107,-
0.408248,0.577350][0.707107,0.408248,-0.577350][0.0,0.816497,0.577350]
    - apply fixed rotation setting to oligomer 2: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <4*e,0,0>
      - apply shift to oligomer 2: <3*e,3*e,3*e>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# P432:{C3}{D4}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <0,0,e>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# p622:{C3}{D6}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,0.57735*e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# F23:{C3}{T}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,0,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates
```

```
########################################

# F432:{C3}{O}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,0,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

########################################

# p4:{C4}{C4}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
        - loop over value of shift parameter e
          - apply shift to oligomer 2: <e,e,0>
          - test coordinates of the oligomers for a designable contact interaction &
write out candidates

########################################

# P432:{C4}{C4}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over z rotations for oligomer 2
      - loop over z translations for oligomer 2
        - apply fixed rotation setting to oligomer 2: [0.0,0.0,1.0][0.0,1.0,0.0][-
1.0,0.0,0.0]
        - loop over value of shift parameter e
          - apply shift to oligomer 2: <0,e,e>
          - test coordinates of the oligomers for a designable contact interaction &
write out candidates

########################################

# p422:{C4}{D2}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <e,0,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# p4212:{C4}{D2}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2: [0.707107,0.707107,0.0][-
0.707107,0.707107,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,0,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# I432:{C4}{D2}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.0,1.0][0.0,1.0,0.0][-
1.0,0.0,0.0]
    - apply fixed rotation setting to oligomer 2:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <2*e,e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# F432:{C4}{D2}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2:
[0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <e,0,e>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# I432:{C4}{D3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <e,e,e>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates
```

```
#########################################

# p422:{C4}{D4}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# P432:{C4}{D4}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1: [0.0,0.0,1.0][0.0,1.0,0.0][-
1.0,0.0,0.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <e,e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# F432:{C4}{T}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <e,e,e>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# P432:{C4}{O}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 1: <e,e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# p622:{C6}{D2}
```

```
- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <e,0,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# p622:{C6}{D3}

- loop over z rotations for oligomer 1
  - loop over z translations for oligomer 1
    - apply fixed rotation setting to oligomer 1:
[1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
    - apply fixed rotation setting to oligomer 2: [0.0,-
1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0]
    - loop over value of shift parameter e
      - apply shift to oligomer 2: <e,0.57735*e,0>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# p222:{D2}{D2}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - loop over value of shift parameter f
    - apply shift to oligomer 2: <e,f,0>
    - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# F222:{D2}{D2}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - loop over value of shift parameter f
    - loop over value of shift parameter g
      - apply shift to oligomer 2: <e,f,g>
      - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# P4222:{D2}{D2}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [0.707107,0.707107,0.0][-
0.707107,0.707107,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - loop over value of shift parameter f
    - apply shift to oligomer 1: <e,0,0>
    - apply shift to oligomer 2: <0,0,f>
    - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# P6222:{D2}{D2}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [0.5,-
0.866025,0.0][0.866025,0.5,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - loop over value of shift parameter f
    - apply shift to oligomer 1: <e,0,0>
    - apply shift to oligomer 2: <0,0,-f>
    - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# P4232:{D2}{D2}

- apply fixed rotation setting to oligomer 1: [0.707107,0.0,0.707107][0.0,1.0,0.0][-
0.707107,0.0,0.707107]
- apply fixed rotation setting to oligomer 2: [0.707107,0.707107,0.0][-
0.707107,0.707107,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <0,e,2*e>
  - apply shift to oligomer 2: <0,2*e,e>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

##########################################

# p622:{D2}{D3}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [0.0,-1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <e,0,0>
  - apply shift to oligomer 2: <e,0.57735*e,0>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

##########################################

# P622:{D2}{D3}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [0.0,-1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - loop over value of shift parameter f
    - apply shift to oligomer 1: <e,0,0>
    - apply shift to oligomer 2: <e,0.57735*e,f>
    - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# P4232:{D2}{D3}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
```

```
- loop over value of shift parameter e
  - apply shift to oligomer 1: <0,0,2*e>
  - apply shift to oligomer 2: <e,e,e>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

##########################################

# I4132:{D2}{D3}

- apply fixed rotation setting to oligomer 1: [0.707107,0.0,0.707107][0.0,1.0,0.0][-
0.707107,0.0,0.707107]
- apply fixed rotation setting to oligomer 2: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <2*e,e,0>
  - apply shift to oligomer 2: <e,e,e>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

##########################################

# p422:{D2}{D4}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <e,0,0>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

##########################################

# P422:{D2}{D4}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - loop over value of shift parameter f
    - apply shift to oligomer 1: <e,0,f>
    - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# I422:{D2}{D4}

- apply fixed rotation setting to oligomer 1: [0.707107,0.707107,0.0][-
0.707107,0.707107,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - loop over value of shift parameter f
    - apply shift to oligomer 1: <e,0,f>
    - test coordinates of the oligomers for a designable contact interaction & write
out candidates

##########################################

# I432:{D2}{D4}

- apply fixed rotation setting to oligomer 1: [0.707107,0.0,0.707107][0.0,1.0,0.0][-
0.707107,0.0,0.707107]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <0,e,2*e>
  - apply shift to oligomer 2: <0,0,2*e>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################

# p622:{D2}{D6}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <e,0,0>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################

# P622:{D2}{D6}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - loop over value of shift parameter f
    - apply shift to oligomer 1: <e,0,f>
    - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# P23:{D2}{T}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <e,0,0>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################

# P23:{D2}{T}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <e,e,0>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################

# F432:{D2}{T}

- apply fixed rotation setting to oligomer 1: [0.707107,0.0,0.707107][0.0,1.0,0.0][-
0.707107,0.0,0.707107]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <e,0,e>
  - apply shift to oligomer 2: <e,e,e>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates
```

```
########################################

# P4232:{D2}{T}

- apply fixed rotation setting to oligomer 1: [0.707107,0.0,0.707107][0.0,1.0,0.0][-
0.707107,0.0,0.707107]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <2*e,e,0>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

########################################

# F432:{D2}{O}

- apply fixed rotation setting to oligomer 1: [0.707107,0.0,0.707107][0.0,1.0,0.0][-
0.707107,0.0,0.707107]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <e,0,e>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

########################################

# I432:{D2}{O}

- apply fixed rotation setting to oligomer 1: [0.707107,0.0,0.707107][0.0,1.0,0.0][-
0.707107,0.0,0.707107]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <2*e,e,0>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

########################################

# p312:{D3}{D3}

- apply fixed rotation setting to oligomer 1: [0.0,-1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [0.0,-1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 2: <e,0.57735*e,0>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

########################################

# P312:{D3}{D3}

- apply fixed rotation setting to oligomer 1: [0.0,-1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [0.0,-1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - loop over value of shift parameter f
    - apply shift to oligomer 2: <e,0.57735*e,f>
    - test coordinates of the oligomers for a designable contact interaction & write
out candidates

########################################

# P6322:{D3}{D3}
```

```
- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [0.0,-1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - loop over value of shift parameter f
    - apply shift to oligomer 2: <e,0.57735*e,f>
    - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# P4232:{D3}{D3}

- apply fixed rotation setting to oligomer 1: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
- apply fixed rotation setting to oligomer 2: [0.707107,-
0.408248,0.577350][0.707107,0.408248,-0.577350][0.0,0.816497,0.577350]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <e,e,e>
  - apply shift to oligomer 2: <e,3*e,e>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################

# P4132:{D3}{D3}

- apply fixed rotation setting to oligomer 1: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
- apply fixed rotation setting to oligomer 2: [0.707107,-
0.408248,0.577350][0.707107,0.408248,-0.577350][0.0,0.816497,0.577350]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <3*e,3*e,3*e>
  - apply shift to oligomer 2: <e,3*e,5*e>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################

# I432:{D3}{D4}

- apply fixed rotation setting to oligomer 1: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <e,e,e>
  - apply shift to oligomer 2: <0,0,2*e>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################

# p622:{D3}{D6}

- apply fixed rotation setting to oligomer 1: [0.0,-1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <e,0.57735*e,0>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################
```

```
# P622:{D3}{D6}

- apply fixed rotation setting to oligomer 1: [0.0,-1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - loop over value of shift parameter f
    - apply shift to oligomer 1: <e,0.57735*e,f>
    - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# F4132:{D3}{T}

- apply fixed rotation setting to oligomer 1: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <e,e,e>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################

# I432:{D3}{O}

- apply fixed rotation setting to oligomer 1: [0.707107,0.408248,0.577350][-
0.707107,0.408248,0.577350][0.0,-0.816497,0.577350]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <e,e,e>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################

# p422:{D4}{D4}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 2: <e,e,0>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################

# P422:{D4}{D4}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - loop over value of shift parameter f
    - apply shift to oligomer 2: <e,e,f>
    - test coordinates of the oligomers for a designable contact interaction & write
out candidates

#########################################

# P432:{D4}{D4}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [0.0,0.0,1.0][0.0,1.0,0.0][-1.0,0.0,0.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <0,0,e>
  - apply shift to oligomer 2: <0,e,e>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################

# P432:{D4}{O}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <0,0,e>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################

# P432:{D4}{O}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <e,e,0>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################

# F23:{T}{T}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 2: <e,e,e>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################

# F23:{T}{T}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 2: <e,0,0>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################

# F432:{T}{O}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 1: <e,e,e>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

#########################################
```

```
# P432:{O}{O}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 2: <e,e,e>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

########################################

# F432:{O}{O}

- apply fixed rotation setting to oligomer 1: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- apply fixed rotation setting to oligomer 2: [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0]
- loop over value of shift parameter e
  - apply shift to oligomer 2: <e,0,0>
  - test coordinates of the oligomers for a designable contact interaction & write out
candidates

########################################
```

## Entanglement and implausible symmetry combinations

The text describes one example of the simplest type of entanglement problem, wherein the combination of symmetries elevates the symmetry of the Wykoff positions on which the individual components sit (Supplemental Figure S1A).

A distinct type of problem occurs in multiple contexts, from finite cage symmetries to extended two and three-dimensional arrays. The simplest example is exemplified by an attempt to combine two different 4-fold tetrameric units together at a 90° angle. Two 4-fold symmetry axes, perpendicular and intersecting, generate octahedral point symmetry O (i.e. 24 molecular orientations with a cubic shape). But a physical construction based on contact between the two different oligomeric types is problematic. According to symmetry O, all six of the cubic faces would have to be occupied by oligomers of the first type, and also by the second type. Immediate collision between the different oligomer types might be avoided by placing one type more central to the overall structure and the other type more peripherally, but the resulting molecular interfaces between the components would then have to be intricately intertwined to avoid collision. Problems of this type arise in cases where the symmetry axes of the separate components become equivalent (i.e. interchangeable) under the resulting combined symmetry. In the problematic case of O:{C4}{C4}, the perpendicular 4-fold axes of the component oligomers are indistinguishable (in position and direction) in symmetry O. Another case, occurring in two-dimensional layers, arises in an attempt to generate layer symmetry p6 from two C6 hexameric units displaced laterally, p6:{C6}{C6} (Figure S1B). Analyzing p6 layer symmetry one sees that all the instances of 6-fold symmetry axes are related by unit cell translation. Therefore, two C6 oligomers will generate p6, with both oligomer types needing to occupy each 6-fold axis of symmetry. This is impossible for compact shapes, but entwined shapes show that the result is mathematically allowable (Figure S1B)(10). Another example arises in an attempted construction of a 2-dimensional layer by combining two different D2 symmetric tetrameric units, rotated 30 degree relative to each other and displaced laterally. The layer symmetry group product is p622, but (p622:{D2}{D2}) is impossible without entwinement. The same is true for combining two different D2 tetramers to construct a 3-dimensional crystal with I4132 symmetry (I4132:{D2}{D2}). In these cases, the two oligomeric components sit on Wykoff positions that are symmetry related. Numerous problematic cases in this category involve two components with the same point group symmetry, but this is not disallowed generally; the SCM table shows many plausible constructions based on components with the same symmetry.

A third, even more subtle problem type was identified in a smaller number of cases that appeared at first to be allowable for compact shapes. The underlying idea for SCMs is that oligomer type 1, suitably oriented and shifted, needs to contact oligomer type 2. Among some seemingly plausible constructions, a situation arises where oligomer 1 (as a compact shape) cannot reach *the necessary instance* of oligomer 2 without colliding with an *unintended* copy of oligomer 2, i.e. one with whom contact would generate a different SCM. These cases could generally be understood by considering the distances between Wykoff positions of different types, looking for whether the distances between the *intended* copies of oligomer 1 and oligomer 2 were always greater than distances between copies that would make unintended contact. Problems in this category were revealed by manual inspection and by our computational exercise to prove constructability. A notable case was an attempted construction of I432:{C4}{D2}. This construction could not be realized, even when a degree of elongation was permitted in the model subunits; e.g. each monomer composed of two spheres instead of one.

The cases of mathematically legal symmetry combinations which are not permitted for compact shapes is vast, and sometimes non-obvious. In fact, there are many construction types where indefinite numbers of entwined variations can be invented based on alternate translational choices for the components. Further exploration would be required to characterize the deep space of mathematically possible, but practically implausible, interwoven constructions. The present study focuses on the design space possible with compact, non-entagled, molecular shapes. This motivated our computational studies in which we validated the constructability of all the entries within the allowable space of SCMs (Text Table 1, Table S3, Figures S2 to S4).

## Choosing Component Oligomers for Example SCMs Constructions

The following steps were taken for the purposes of illustrating candidate SCM constructions in Figure 4. The advanced search tool in the Protein Data Bank was used to obtain sets of C3, D4, C4 and D3 protein homo-oligomers clustered at 70% sequence identity with: 1) X-ray resolution less than 2.5 Å, 2) *Escherichia coli* as the organism used for protein expression and 3) at least 30% alpha helical. Membrane proteins were removed. Biological assemblies were identified using QSBio (10) and were then downloaded from the Protein Data Bank. A C3 trimer and a D4 octamer were arbitrarily chosen from the sets of curated structures as component oligomers for the P432 crystal presented in Figure 4. Similarly, a C4 tetramer and a D3 hexamer were arbitrarily selected from the sets of curated structures to construct the I432 crystal shown in Figure 4. Other selection criteria for choosing component oligomers are of course possible.

**Comparison to MOF Networks**

In the language of MOFs, the protein-based SCMs discussed here would be described as *binodal* nets with *[2,1]* transitivity, meaning two different types of nodes (one corresponding to each type of oligomer) and one type of edge, all edges being identical to each other under symmetry, each edge connecting nodes of alternate types. Thousands of MOF compounds have been structurally characterized to date, and as a result a great many distinct three-dimensional ('3-periodic') nets have been identified and categorized (11). Some 73 types of binodal MOF nets have been described (understood to not be a complete set) but only five of those are chiral (i.e. lacking mirror or center of inversion operations) and can be mapped to table entries for our protein SCMs. [N.B. The vast majority of MOF materials have achiral centers, often individual metal atom sites.] Some of the symmetric architectures we lay out for protein materials have relatives among the bimodal MOF nets that have been described, e.g. being interconvertible by addition of a center of inversion at an appropriate position, but others appear to be more clearly distinct from any that have been described before. Further comparisons between MOF materials and chiral protein-based SCMs could find fertile ground. Two additional points of distinction compared to exploration of MOF networks are: (1) our multiplication table articulates *all* possible binodal [2,1]-transitive architectures and their key properties (subject to construction from compact molecules), and (2) the growing facility with which protein molecules can be re-engineered to bind to each other in precisely defined configurations should enable the exploration of specific and predictable materials outcomes.

**SUPPLEMENTARY TABLES**

**Table S1.** Point group combination possibilities, with orientation specifications and outcomes.

| **Resulting point group** | **Component point groups and orientational settings**[*] |
|---|---|
| C3 | {C3[#1]}{C3[#1]} |
| C4 | {C2[#1]}{C4[#1]}; {C4[#1]}{C4[#1]} |
| C6 | {C2[#1]}{C3[#1]}; {C2[#1]}{C6[#1]}; {C3[#1]}{C6[#1]} |
| D2 | {C2[#1]}{D2[#1]}; {D2[#1]}{D2[#1]} |
| D3 | {C2[#2]}{C3[#1]}; {C2[#6]}{C3[#1]}; {C2[#6]}{D3[#11]}; {C2[#2]}{D3[#1]}; {C3[#1]}{D3[#11]}; {C3[#1]}{D3[#1]}; {D3[#11]}{D3[#11]} |
| D4 | {C2[#2]}{C4[#1]}; {C2[#8]}{C4[#1]}; {C2[#8]}{D2[#1]}; {C2[#2]}{D2[#5]}; {C2[#1]}{D4[#1]}; {C2[#2]}{D4[#1]}; {C2[#8]}{D4[#1]}; {C4[#1]}{D2[#1]}; {C4[#1]}{D2[#5]}; {C4[#1]}{D4[#1]}; {D2[#1]}{D2[#5]}; {D2[#1]}{D4[#1]}; {D2[#5]}{D4[#1]}; {D4[#1]}{D4[#1]} |
| D5 | {C2[#2]}{C5[#1]} |
| D6 | {C2[#2]}{C6[#1]}; {C2[#6]}{C6[#1]}; {C2[#10]}{D2[#1]}; {C2[#1]}{D3[#11]}; {C2[#2]}{D3[#11]}; {C2[#1]}{D6[#1]}; {C2[#6]}{D6[#1]}; {C3[#1]}{D2[#1]}; {C3[#1]}{D6[#1]}; {C6[#1]}{D2[#1]}; {C6[#1]}{D3[#11]}; {D2[#1]}{D2[#13]}; {D2[#1]}{D3[#11]}; {D2[#1]}{D3[#4]}; {D2[#1]}{D6[#1]}; {D3[#1]}{D3[#11]}; {D3[#11]}{D6[#1]} |
| T | {C2[#1]}{C3[#4]}; {C2[#1]}{T[#1]}; {C3[#4]}{C3[#12]}; {C3[#4]}{D2[#1]}; {C3[#4]}{T[#1]}; {D2[#1]}{T[#1]}; {T[#1]}{T[#1]} |
| O | {C2[#3]}{C3[#4]}; {C2[#3]}{C4[#1]}; {C2[#3]}{D2[#5]}; {C2[#1]}{D3[#4]}; {C2[#3]}{D3[#4]}; {C2[#3]}{D4[#1]}; {C2[#3]}{T[#1]}; {C2[#1]}{O[#1]}; {C2[#3]}{O[#1]}; {C3[#4]}{C4[#1]}; {C3[#4]}{D2[#3]}; {C3[#12]}{D3[#4]}; {C3[#4]}{D4[#1]}; {C3[#4]}{O[#1]}; {C4[#1]}{C4[#2]}; {C4[#1]}{D2[#3]}; {C4[#2]}{D2[#3]}; {C4[#1]}{D3[#4]}; {C4[#2]}{D4[#1]}; {C4[#1]}{T[#1]}; {C4[#1]}{O[#1]}; {D2[#3]}{D2[#5]}; {D2[#3]}{D3[#4]}; {D2[#3]}{D4[#1]}; {D2[#3]}{T[#1]}; {D2[#3]}{O[#1]}; {D3[#4]}{D3[#12]}; {D3[#4]}{D4[#1]}; {D3[#4]}{T[#1]}; {D3[#4]}{O[#1]}; {D4[#1]}{D4[#2]}; {D4[#1]}{O[#1]}; {T[#1]}{O[#1]}; {O[#1]}{O[#1]} |
| I | {C2[#1]}{C3[#7]}; {C2[#1]}{C5[#9]}; {C3[#7]}{C5[#9]} |
| | |
| **Setting #** | **Setting orientation matrices** [$] |
| 1 | [1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,1.0] |
| 2 | [0.0,0.0,1.0][0.0,1.0,0.0][-1.0,0.0,0.0] |
| 3 | [0.707107,0.0,0.707107][0.0,1.0,0.0][-0.707107,0.0,0.707107] |
| 4 | [0.707107,0.408248,0.577350][-0.707107,0.408248,0.577350][0.0,-0.816497,0.577350] |
| 5 | [0.707107,0.707107,0.0][-0.707107,0.707107,0.0][0.0,0.0,1.0] |
| 6 | [1.0,0.0,0.0][0.0,0.0,1.0][0.0,-1.0,0.0] |
| 7 | [1.0,0.0,0.0][0.0,0.934172,0.356822][0.0,-0.356822,0.934172] |

| | |
|---|---|
| 8 | [0.0,0.707107,0.707107][0.0,-0.707107,0.707107][1.0,0.0,0.0] |
| 9 | [0.850651,0.0,0.525732][0.0,1.0,0.0][-0.525732,0.0,0.850651] |
| 10 | [0.0,0.5,0.866025][0.0,-0.866025,0.5][1.0,0.0,0.0] |
| 11 | [0.0,-1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0] |
| 12 | [0.707107,-0.408248,0.577350][0.707107,0.408248,-0.577350][0.0,0.816497,0.577350] |
| 13 | [0.5,-0.866025,0.0][0.866025,0.5,0.0][0.0,0.0,1.0] |

*Setting numbers are specified in square brackets in the top section as they would be applied to their respective component point groups, beginning in their canonical orientations. Canonical orientations are with the unique (highest symmetry) axis along z, with the exception that symmetry T is set with its underlying D2 symmetry along x, y, and z. In D3, a 2-fold axis is set along x.

$Rotation matrices in the bottom section are written in rows, with matrix multiplication to occur on the left, as in multiplying column coordinate vectors.

**Table S2.** Orientational Degeneracies.

| Component Point Group | Degeneracy Matrices | Description |
|---|---|---|
| **Cn** | [-1.0,0.0,0.0][0.0,1.0,0.0][0.0,0.0,-1.0] | 180° rotation about y |
| **D2** | [0.0,0.0,1.0][1.0,0.0,0.0][0.0,1.0,0.0]<br>[0.0,1.0,0.0][0.0,0.0,1.0][1.0,0.0,0.0]<br>[-1.0,0.0,0.0][0.0,0.0,1.0][0.0,1.0,0.0]<br>[0.0,0.0,1.0][0.0,-1.0,0.0][1.0,0.0,0.0]<br>[0.0,1.0,0.0][1.0,0.0,0.0][0.0,0.0,-1.0] | z, x, y<br>y, z, x<br>-x, z, y<br>z, -y, x<br>y, x, -z |
| **D3** | [0.5,-0.86603,0.0][0.86603,0.5,0.0][0.0,0.0,1.0] | 60° rotation about z |
| **D4** | [0.707107,0.707107,0.0][-<br>0.707107,0.707107,0.0][0.0,0.0,1.0] | 45° rotation about z |
| **D6** | [0.86603,-0.5,0.0][0.5,0.86603,0.0][0.0,0.0,1.0] | 30° rotation about z |
| **T** | [0.0,-1.0,0.0][1.0,0.0,0.0][0.0,0.0,1.0] | 90° rotation about z |

For a given point group symmetry, this table describes the set of rotations (in addition to axial rotations for cases of cyclic symmetry) that need to be considered for a protein oligomer of that symmetry when performing a symmetry-based docking search in order to sample the full space of allowable orientations. The problem of degeneracies here relates closely to two well-known problems in crystallography: Cheshire symmetry groups for analyzing equivalent origins, and merohedral twinning operations. Briefly, in the present case we must analyze rotations that are not part of the underlying symmetry in question, but which leave the symmetry elements themselves unaffected. For example, symmetry D2 has 2-fold axes of symmetry along three perpendicular directions. A rotation by 90 degrees about any axis is not an element of D2 symmetry, yet such an operation results in an indistinguishable configuration of underlying symmetry elements. Those operations (and others) therefore need to be considered when sampling the allowable orientation space for a component having D2 symmetry. A precise mathematical description of the solution is as follows. If ***G*** is the underlying point group, let ***H*** be the supergroup of ***G*** such that for any element ***h*** in ***H***, ***G*** is unaffected by a similarity transformation by ***h***, meaning $\boldsymbol{hGh^{-1}} = \boldsymbol{G}$. The required orientation degeneracy matrices are a *traversal* (of order $|\boldsymbol{H}|/|\boldsymbol{G}|$) of the *cosets* of ***G*** in ***H***. In the table above, the identity orientation matrix is not listed but is understood. Each orientation matrix is written as a set of rows, though the transpositional sense of the matrices (rows vs columns) is unimportant here. The matrices are written out in reference frames consistent with their application to component point groups in their canonical orientations (described in Table S1).

**Table S3.** Master Table of SCMs

| Entry # | Group 1 | Internal dof (group 1) | Rotational setting (group 1) | Translation dof (group 1) | Group 2 | Internal dof (group 2) | Rotational setting (group 2) | Translation dof (group 2) | RESULT | Unit cell | Total degrees of freedom | Ring size |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | C2 | r:<0,0,1,a> t:<0,0,b> | 2 | <0,0,0> | C2 | r:<0,0,1,c> t:<0,0,d> | 1 | <0,0,0> | D2 | N/A | 4 | 2 |
| 2 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,0,0> | C3 | r:<0,0,1,c> | 1 | <e,0.577350*e, 0> | p6 | (2*e, 2*e), 120 | 4 | 6 |
| 3 | C2 | r:<0,0,1,a> t:<0,0,b> | 2 | <0,0,0> | C3 | r:<0,0,1,c> t:<0,0,d> | 1 | <0,0,0> | D3 | N/A | 4 | 2 |
| 4 | C2 | r:<0,0,1,a> t:<0,0,b> | 6 | <e,0,0> | C3 | r:<0,0,1,c> t:<0,0,d> | 1 | <0,0,0> | p312 | (2*e, 2*e), 120 | 5 | 6 |
| 5 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <0,0,0> | C3 | r:<0,0,1,c> t:<0,0,d> | 4 | <0,0,0> | T | N/A | 4 | 3 |
| 6 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <0,e,0> | C3 | r:<0,0,1,c> t:<0,0,d> | 4 | <0,0,0> | I213 | (4*e, 4*e, 4*e), (90, 90, 90) | 5 | 10 |
| 7 | C2 | r:<0,0,1,a> t:<0,0,b> | 3 | <0,0,0> | C3 | r:<0,0,1,c> t:<0,0,d> | 4 | <0,0,0> | O | N/A | 4 | 4 |
| 8 | C2 | r:<0,0,1,a> t:<0,0,b> | 3 | <2*e,e,0> | C3 | r:<0,0,1,c> t:<0,0,d> | 4 | <0,0,0> | P4132 | (8*e, 8*e, 8*e), (90, 90, 90) | 5 | 10 |
| 9 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <0,0,0> | C3 | r:<0,0,1,c> t:<0,0,d> | 7 | <0,0,0> | I | N/A | 4 | 5 |
| 10 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,0,0> | C4 | r:<0,0,1,c> | 1 | <0,0,0> | p4 | (2*e, 2*e), 90 | 4 | 4 |
| 11 | C2 | r:<0,0,1,a> t:<0,0,b> | 2 | <0,0,0> | C4 | r:<0,0,1,c> t:<0,0,d> | 1 | <0,0,0> | D4 | N/A | 4 | 2 |
| 12 | C2 | r:<0,0,1,a> t:<0,0,b> | 8 | <0,0,0> | C4 | r:<0,0,1,c> t:<0,0,d> | 1 | <e,0,0> | p4212 | (2*e, 2*e), 90 | 5 | 4 |
| 13 | C2 | r:<0,0,1,a> t:<0,0,b> | 3 | <0,0,0> | C4 | r:<0,0,1,c> t:<0,0,d> | 1 | <0,0,0> | O | N/A | 4 | 3 |
| 14 | C2 | r:<0,0,1,a> t:<0,0,b> | 3 | <2*e,e,0> | C4 | r:<0,0,1,c> t:<0,0,d> | 1 | <0,0,0> | I432 | (4*e, 4*e, 4*e), (90, 90, 90) | 5 | 8 |
| 15 | C2 | r:<0,0,1,a> t:<0,0,b> | 2 | <0,0,0> | C5 | r:<0,0,1,c> t:<0,0,d> | 1 | <0,0,0> | D5 | N/A | 4 | 2 |
| 16 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <0,0,0> | C5 | r:<0,0,1,c> t:<0,0,d> | 9 | <0,0,0> | I | N/A | 4 | 3 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 17 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,0,0> | C6 | r:<0,0,1,c> | 1 | <0,0,0> | p6 | (2*e, 2*e), 120 | 4 | 3 |
| 18 | C2 | r:<0,0,1,a> t:<0,0,b> | 2 | <0,0,0> | C6 | r:<0,0,1,c> t:<0,0,d> | 1 | <0,0,0> | D6 | N/A | 4 | 2 |
| 19 | C2 | r:<0,0,1,a> t:<0,0,b> | 6 | <e,0,0> | C6 | r:<0,0,1,c> t:<0,0,d> | 1 | <0,0,0> | p622 | (2*e, 2*e), 120 | 5 | 4 |
| 20 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,f,0> | D2 | [redacted] | 1 | <0,0,0> | c222 | (4*e, 4*f), 90 | 4 | 4 |
| 21 | C2 | r:<0,0,1,a> t:<0,0,b> | 8 | <0,0,0> | D2 | [redacted] | 1 | <e,0,0> | p422 | (2*e, 2*e), 90 | 3 | 4 |
| 22 | C2 | r:<0,0,1,a> t:<0,0,b> | 2 | <0,e,f> | D2 | [redacted] | 5 | <0,0,0> | I4122 | (4*e, 4*e, 8*f), (90, 90, 90) | 4 | 6 |
| 23 | C2 | r:<0,0,1,a> t:<0,0,b> | 10 | <0,0,0> | D2 | [redacted] | 1 | <e,0,0> | p622 | (2*e, 2*e), 120 | 3 | 3 |
| 24 | C2 | r:<0,0,1,a> t:<0,0,b> | 10 | <0,0,e> | D2 | [redacted] | 1 | <f,0,0> | P6222 | (2*f, 2*f, 6*e), (90, 90, 120) | 4 | 6 |
| 25 | C2 | r:<0,0,1,a> t:<0,0,b> | 3 | <0,0,0> | D2 | [redacted] | 5 | <2*e,0,e> | I432 | (4*e, 4*e, 4*e), (90, 90, 90) | 3 | 4 |
| 26 | C2 | r:<0,0,1,a> t:<0,0,b> | 3 | <-2*e,3*e,0> | D2 | [redacted] | 5 | <0,2*e,e> | I4132 | (8*e, 8*e, 8*e), (90, 90, 90) | 3 | 3 |
| 27 | C2 | r:<0,0,1,a> t:<0,0,b> | 6 | <e,0,0> | D3 | [redacted] | 11 | <0,0,0> | p312 | (2*e, 2*e), 120 | 3 | 3 |
| 28 | C2 | r:<0,0,1,a> t:<0,0,b> | 2 | <0,e,f> | D3 | [redacted] | 1 | <0,0,0> | R32 | (3.4641*e , 3.4641*e, 3*f), (90, 90, 120) | 4 | 4 |
| 29 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,0,0> | D3 | [redacted] | 11 | <e,0.57735*e,0 > | p622 (a) | (2*e, 2*e), 120 | 3 | 2 |
| 30 | C2 | r:<0,0,1,a> t:<0,0,b> | 2 | <0,0,0> | D3 | [redacted] | 11 | <e,0.57735*e,0 > | p622 (b) | (2*e, 2*e), 120 | 3 | 2 |
| 31 | C2 | r:<0,0,1,a> t:<0,0,b> | 2 | <0,0,0> | D3 | [redacted] | 11 | <e,0.57735*e,f> | P6322 | (2*e, 2*e, 4*f), (90, 90, 120) | 4 | 4 |
| 32 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <0,0,0> | D3 | [redacted] | 4 | <e,e,e> | F4132 | (8*e, 8*e, 8*e), (90, 90, 90) | 3 | 3 |
| 33 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <0,2*e,0> | D3 | [redacted] | 4 | <e,e,e> | I4132 (a) | (8*e, 8*e, 8*e), (90, 90, 90) | 3 | 2 |
| 34 | C2 | r:<0,0,1,a> t:<0,0,b> | 3 | <0,0,0> | D3 | [redacted] | 4 | <e,e,e> | I432 | (4*e, 4*e, 4*e), (90, 90, 90) | 3 | 4 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 35 | C2 | r:<0,0,1,a> t:<0,0,b> | 3 | <0,e,-2*e> | D3 | [redacted] | 4 | <e,e,e> | I4132 (b) | (8*e, 8*e, 8*e), (90, 90, 90) | 3 | 2 |
| 36 | C2 | r:<0,0,1,a> t:<0,0,b> | 3 | <0,e,-2*e> | D3 | [redacted] | 4 | <3*e,3*e,3*e> | P4132 | (8*e, 8*e, 8*e), (90, 90, 90) | 3 | 3 |
| 37 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,0,0> | D4 | [redacted] | 1 | <0,0,0> | p422 (a) | (2*e, 2*e), 90 | 3 | 2 |
| 38 | C2 | r:<0,0,1,a> t:<0,0,b> | 2 | <0,e,0> | D4 | [redacted] | 1 | <0,0,0> | p422 (b) | (2*e, 2*e), 90 | 3 | 2 |
| 39 | C2 | r:<0,0,1,a> t:<0,0,b> | 8 | <0,e,f> | D4 | [redacted] | 1 | <0,0,0> | I422 | (2*e, 2*e, 4*f), (90, 90, 90) | 4 | 4 |
| 40 | C2 | r:<0,0,1,a> t:<0,0,b> | 3 | <0,0,0> | D4 | [redacted] | 1 | <0,0,e> | P432 | (2*e, 2*e, 2*e), (90, 90, 90) | 3 | 3 |
| 41 | C2 | r:<0,0,1,a> t:<0,0,b> | 3 | <2*e,e,0> | D4 | [redacted] | 1 | <2*e,2*e,0> | I432 | (4*e, 4*e, 4*e), (90, 90, 90) | 3 | 2 |
| 42 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,0,0> | D6 | [redacted] | 1 | <0,0,0> | p622 (a) | (2*e, 2*e), 120 | 3 | 2 |
| 43 | C2 | r:<0,0,1,a> t:<0,0,b> | 6 | <e,0,0> | D6 | [redacted] | 1 | <0,0,0> | p622 (b) | (2*e, 2*e), 120 | 3 | 2 |
| 44 | C2 | r:<0,0,1,a> t:<0,0,b> | 6 | <e,0,f> | D6 | [redacted] | 1 | <0,0,0> | P622 | (2*e, 2*e, 2*f), (90, 90, 120) | 4 | 4 |
| 45 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,0,0> | T | [redacted] | 1 | <0,0,0> | P23 | (2*e, 2*e, 2*e), (90, 90, 90) | 3 | 2 |
| 46 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,e,0> | T | [redacted] | 1 | <0,0,0> | F23 | (4*e, 4*e, 4*e), (90, 90, 90) | 3 | 3 |
| 47 | C2 | r:<0,0,1,a> t:<0,0,b> | 3 | <2*e,3*e,0> | T | [redacted] | 1 | <0,4*e,0> | F4132 | (8*e, 8*e, 8*e), (90, 90, 90) | 3 | 2 |
| 48 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,0,0> | O | [redacted] | 1 | <0,0,0> | P432 (a) | (2*e, 2*e, 2*e), (90, 90, 90) | 3 | 2 |
| 49 | C2 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,e,0> | O | [redacted] | 1 | <0,0,0> | F432 (a) | (4*e, 4*e, 4*e), (90, 90, 90) | 3 | 2 |
| 50 | C2 | r:<0,0,1,a> t:<0,0,b> | 3 | <e,0,0> | O | [redacted] | 1 | <0,0,0> | F432 (b) | (2*e, 2*e, 2*e), (90, 90, 90) | 3 | 2 |
| 51 | C2 | r:<0,0,1,a> t:<0,0,b> | 3 | <0,e,0> | O | [redacted] | 1 | <0,0,0> | P432 (b) | (2*e, 2*e, 2*e), (90, 90, 90) | 3 | 2 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 52 | C2 | r:<0,0,1,a> t:<0,0,b> | 3 | <-e,e,e> | O | ■ | 1 | <0,0,0> | I432 | (4*e, 4*e, 4*e), (90, 90, 90) | 3 | 2 |
| 53 | C3 | r:<0,0,1,a> t:<0,0,b> | 1 | <0,0,0> | C3 | r:<0,0,1,c> | 1 | <e,0.57735*e,0 > | p3 | (2*e, 2*e), 120 | 4 | 3 |
| 54 | C3 | r:<0,0,1,a> t:<0,0,b> | 4 | <0,0,0> | C3 | r:<0,0,1,c> t:<0,0,d> | 12 | <0,0,0> | T | N/A | 4 | 2 |
| 55 | C3 | r:<0,0,1,a> t:<0,0,b> | 4 | <0,0,0> | C3 | r:<0,0,1,c> t:<0,0,d> | 12 | <e,0,0> | P213 | (2*e, 2*e, 2*e), (90, 90, 90) | 5 | 5 |
| 56 | C3 | r:<0,0,1,a> t:<0,0,b> | 4 | <0,0,0> | C4 | r:<0,0,1,c> t:<0,0,d> | 1 | <0,0,0> | O | N/A | 4 | 2 |
| 57 | C3 | r:<0,0,1,a> t:<0,0,b> | 4 | <0,0,0> | C4 | r:<0,0,1,c> t:<0,0,d> | 1 | <e,0,0> | F432 | (2*e, 2*e, 2*e), (90, 90, 90) | 5 | 6 |
| 58 | C3 | r:<0,0,1,a> t:<0,0,b> | 7 | <0,0,0> | C5 | r:<0,0,1,c> t:<0,0,d> | 9 | <0,0,0> | I | N/A | 4 | 2 |
| 59 | C3 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,0.57735*e ,0> | C6 | r:<0,0,1,c> | 1 | <0,0,0> | p6 | (2*e, 2*e), 120 | 4 | 2 |
| 60 | C3 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,0.57735*e ,0> | D2 | ■ | 1 | <e,0,0> | p622 | (2*e, 2*e), 120 | 3 | 2 |
| 61 | C3 | r:<0,0,1,a> t:<0,0,b> | 4 | <0,0,0> | D2 | ■ | 1 | <e,0,0> | P23 | (2*e, 2*e, 2*e), (90, 90, 90) | 3 | 3 |
| 62 | C3 | r:<0,0,1,a> t:<0,0,b> | 4 | <0,0,0> | D2 | ■ | 3 | <e,0,e> | F432 | (4*e, 4*e, 4*e), (90, 90, 90) | 3 | 3 |
| 63 | C3 | r:<0,0,1,a> t:<0,0,b> | 4 | <0,0,0> | D2 | ■ | 3 | <2*e,e,0> | I4132 | (8*e,8*e, 8*e), (90, 90, 90) | 3 | 2 |
| 64 | C3 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,0.57735*e ,0> | D3 | ■ | 11 | <0,0,0> | p312 | (2*e, 2*e), 120 | 3 | 2 |
| 65 | C3 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,0.57735*e ,0> | D3 | ■ | 1 | <0,0,0> | p321 | (2*e, 2*e), 120 | 3 | 2 |
| 66 | C3 | r:<0,0,1,a> t:<0,0,b> | 12 | <4*e,0,0> | D3 | ■ | 4 | <3*e,3*e,3*e> | P4132 | (8*e, 8*e, 8*e), (90, 90, 90) | 3 | 4 |
| 67 | C3 | r:<0,0,1,a> t:<0,0,b> | 4 | <0,0,0> | D4 | ■ | 1 | <0,0,e> | P432 | (2*e, 2*e, 2*e), (90, 90, 90) | 3 | 2 |
| 68 | C3 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,0.57735*e ,0> | D6 | ■ | 1 | <0,0,0> | p622 | (2*e, 2*e), 120 | 3 | 2 |
| 69 | C3 | r:<0,0,1,a> t:<0,0,b> | 4 | <e,0,0> | T | ■ | 1 | <0,0,0> | F23 | (2*e, 2*e, 2*e), (90, 90, 90) | 3 | 2 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 70 | C3 | r:<0,0,1,a> t:<0,0,b> | 4 | <e,0,0> | O | | 1 | <0,0,0> | F432 | (2*e, 2*e, 2*e), (90, 90, 90) | 3 | 2 |
| 71 | C4 | r:<0,0,1,a> t:<0,0,b> | 1 | <0,0,0> | C4 | r:<0,0,1,c> | 1 | <e,e,0> | p4 | (2*e, 2*e), 90 | 4 | 2 |
| 72 | C4 | r:<0,0,1,a> t:<0,0,b> | 1 | <0,0,0> | C4 | r:<0,0,1,c> t:<0,0,d> | 2 | <0,e,e> | P432 | (2*e, 2*e, 2*e), (90, 90, 90) | 5 | 4 |
| 73 | C4 | r:<0,0,1,a> t:<0,0,b> | 1 | <0,0,0> | D2 | | 1 | <e,0,0> | p422 | (2*e, 2*e), 90 | 3 | 2 |
| 74 | C4 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,0,0> | D2 | | 5 | <0,0,0> | p4212 | (2*e, 2*e), 90 | 3 | 2 |
| 75 | C4 | r:<0,0,1,a> t:<0,0,b> | 2 | <0,0,0> | D2 | | 3 | <2*e,e,0> | I432 | (4*e, 4*e, 4*e), (90, 90, 90) | 3 | 2 |
| 76 | C4 | r:<0,0,1,a> t:<0,0,b> | 1 | <0,0,0> | D2 | | 3 | <e,0,e> | F432 | (4*e, 4*e, 4*e), (90, 90, 90) | 3 | 3 |
| 77 | C4 | r:<0,0,1,a> t:<0,0,b> | 1 | <0,0,0> | D3 | | 4 | <e,e,e> | I432 | (4*e, 4*e, 4*e), (90, 90, 90) | 3 | 2 |
| 78 | C4 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,e,0> | D4 | | 1 | <0,0,0> | p422 | (2*e, 2*e), 90 | 3 | 2 |
| 79 | C4 | r:<0,0,1,a> t:<0,0,b> | 2 | <0,0,0> | D4 | | 1 | <e,e,0> | P432 | (2*e, 2*e, 2*e), (90, 90, 90) | 3 | 2 |
| 80 | C4 | r:<0,0,1,a> t:<0,0,b> | 1 | <0,0,0> | T | | 1 | <e,e,e> | F432 | (4*e, 4*e, 4*e), (90, 90, 90) | 3 | 2 |
| 81 | C4 | r:<0,0,1,a> t:<0,0,b> | 1 | <e,e,0> | O | | 1 | <0,0,0> | P432 | (2*e, 2*e, 2*e), (90, 90, 90) | 3 | 2 |
| 82 | C6 | r:<0,0,1,a> t:<0,0,b> | 1 | <0,0,0> | D2 | | 1 | <e,0,0> | p622 | (2*e, 2*e), 120 | 3 | 2 |
| 83 | C6 | r:<0,0,1,a> t:<0,0,b> | 1 | <0,0,0> | D3 | | 11 | <e,0.57735*e,0 > | p622 | (2*e, 2*e), 120 | 2 | 2 |
| 84 | D2 | | 1 | <0,0,0> | D2 | | 1 | <e,f,0> | p222 | (2*e, 2*f), 90 | 2 | 2 |
| 85 | D2 | | 1 | <0,0,0> | D2 | | 1 | <e,f,g> | F222 | (4*e, 4*f, 4*g), (90, 90, 90) | 3 | 3 |
| 86 | D2 | | 1 | <e,0,0> | D2 | | 5 | <0,0,f> | P4222 | (2*e, 2*e, 4*f), (90, 90, 90) | 2 | 2 |
| 87 | D2 | | 1 | <e,0,0> | D2 | | 13 | <0,0,-f> | P6222 | (2*e, 2*e, 6*f), (90, 90, 120) | 2 | 2 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 88 | D2 | [redacted] | 3 | <0,e,2*e> | D2 | [redacted] | 5 | <0,2*e,e> | P4232 | (4*e, 4*e, 4*e), (90, 90, 90) | 1 | 2 |
| 89 | D2 | [redacted] | 1 | <e,0,0> | D3 | [redacted] | 11 | <e,0.57735*e,0> | p622 | (2*e, 2*e), 120 | 1 | 1 |
| 90 | D2 | [redacted] | 1 | <e,0,0> | D3 | [redacted] | 11 | <e,0.57735*e,f> | P622 | (2*e, 2*e, 2*f), (90, 90, 120) | 2 | 2 |
| 91 | D2 | [redacted] | 1 | <0,0,2*e> | D3 | [redacted] | 4 | <e,e,e> | P4232 | (4*e, 4*e, 4*e), (90, 90, 90) | 1 | 2 |
| 92 | D2 | [redacted] | 3 | <2*e,e,0> | D3 | [redacted] | 4 | <e,e,e> | I4132 | (8*e, 8*e, 8*e), (90, 90, 90) | 1 | 1 |
| 93 | D2 | [redacted] | 1 | <e,0,0> | D4 | [redacted] | 1 | <0,0,0> | p422 | (2*e, 2*e), 90 | 1 | 1 |
| 94 | D2 | [redacted] | 1 | <e,0,f> | D4 | [redacted] | 1 | <0,0,0> | P422 | (2*e, 2*e, 2*f), (90, 90,90) | 2 | 2 |
| 95 | D2 | [redacted] | 5 | <e,0,f> | D4 | [redacted] | 1 | <0,0,0> | I422 | (2*e, 2*e, 4*f), (90, 90,90) | 2 | 2 |
| 96 | D2 | [redacted] | 3 | <0,e,2*e> | D4 | [redacted] | 1 | <0,0,2*e> | I432 | (4*e, 4*e, 4*e), (90, 90, 90) | 1 | 1 |
| 97 | D2 | [redacted] | 1 | <e,0,0> | D6 | [redacted] | 1 | <0,0,0> | p622 | (2*e, 2*e), 120 | 1 | 1 |
| 98 | D2 | [redacted] | 1 | <e,0,f> | D6 | [redacted] | 1 | <0,0,0> | P622 | (2*e, 2*e, 2*f), (90, 90, 120) | 2 | 2 |
| 99 | D2 | [redacted] | 1 | <e,0,0> | T | [redacted] | 1 | <0,0,0> | P23 | (2*e, 2*e, 2*e), (90, 90, 90) | 1 | 1 |
| 100 | D2 | [redacted] | 1 | <e,e,0> | T | [redacted] | 1 | <0,0,0> | P23 | (2*e, 2*e, 2*e), (90, 90, 90) | 1 | 2 |
| 101 | D2 | [redacted] | 3 | <e,0,e> | T | [redacted] | 1 | <e,e,e> | F432 | (4*e, 4*e, 4*e), (90, 90, 90) | 1 | 1 |
| 102 | D2 | [redacted] | 3 | <2*e,e,0> | T | [redacted] | 1 | <0,0,0> | P4232 | (4*e, 4*e, 4*e), (90, 90, 90) | 1 | 2 |
| 103 | D2 | [redacted] | 3 | <e,0,e> | O | [redacted] | 1 | <0,0,0> | F432 | (4*e, 4*e, 4*e), (90, 90, 90) | 1 | 1 |
| 104 | D2 | [redacted] | 3 | <2*e,e,0> | O | [redacted] | 1 | <0,0,0> | I432 | (4*e, 4*e, 4*e), (90, 90, 90) | 1 | 2 |

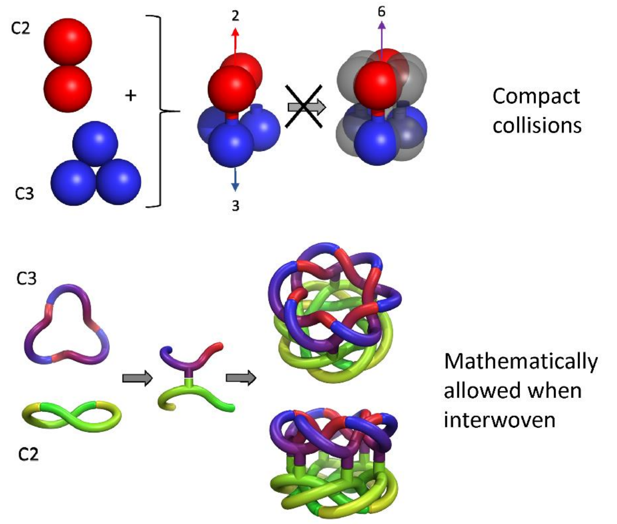
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 105 | D3 | [redacted] | 11 | <0,0,0> | D3 | [redacted] | 11 | <e,0.57735*e,0> | p312 | (2*e, 2*e), 120 | 1 | 1 |
| 106 | D3 | [redacted] | 11 | <0,0,0> | D3 | [redacted] | 11 | <e,0.57735*e,f> | P312 | (2*e, 2*e, 2*f), (90, 90, 120) | 2 | 2 |
| 107 | D3 | [redacted] | 1 | <0,0,0> | D3 | [redacted] | 11 | <e,0.57735*e,f> | P6322 | (2*e, 2*e, 4*f), (90, 90, 120) | 2 | 2 |
| 108 | D3 | [redacted] | 4 | <e,e,e> | D3 | [redacted] | 12 | <e,3*e,e> | P4232 | (4*e, 4*e, 4*e), (90, 90, 90) | 1 | 2 |
| 109 | D3 | [redacted] | 4 | <3*e,3*e,3*e> | D3 | [redacted] | 12 | <e,3*e,5*e> | P4132 | (8*e, 8*e, 8*e), (90, 90, 90) | 1 | 1 |
| 110 | D3 | [redacted] | 4 | <e,e,e> | D4 | [redacted] | 1 | <0,0,2*e> | I432 | (4*e, 4*e, 4*e), (90, 90, 90) | 1 | 2 |
| 111 | D3 | [redacted] | 11 | <e,0.57735*e,0> | D6 | [redacted] | 1 | <0,0,0> | p622 | (2*e, 2*e), 120 | 1 | 1 |
| 112 | D3 | [redacted] | 11 | <e,0.57735*e,f> | D6 | [redacted] | 1 | <0,0,0> | P622 | (2*e, 2*e, 2*f), (90, 90, 120) | 2 | 2 |
| 113 | D3 | [redacted] | 4 | <e,e,e> | T | [redacted] | 1 | <0,0,0> | F4132 | (8*e, 8*e, 8*e), (90, 90, 90) | 1 | 1 |
| 114 | D3 | [redacted] | 4 | <e,e,e> | O | [redacted] | 1 | <0,0,0> | I432 | (4*e, 4*e, 4*e), (90, 90, 90) | 1 | 1 |
| 115 | D4 | [redacted] | 1 | <0,0,0> | D4 | [redacted] | 1 | <e,e,0> | p422 | (2*e, 2*e), 90 | 1 | 1 |
| 116 | D4 | [redacted] | 1 | <0,0,0> | D4 | [redacted] | 1 | <e,e,f> | P422 | (2*e, 2*e, 2*f), (90, 90,90) | 2 | 2 |
| 117 | D4 | [redacted] | 1 | <0,0,e> | D4 | [redacted] | 2 | <0,e,e> | P432 | (2*e, 2*e, 2*e), (90, 90, 90) | 1 | 1 |
| 118 | D4 | [redacted] | 1 | <0,0,e> | O | [redacted] | 1 | <0,0,0> | P432 | (2*e, 2*e, 2*e), (90, 90, 90) | 1 | 1 |
| 119 | D4 | [redacted] | 1 | <e,e,0> | O | [redacted] | 1 | <0,0,0> | P432 | (2*e, 2*e, 2*e), (90, 90, 90) | 1 | 1 |
| 120 | T | [redacted] | 1 | <0,0,0> | T | [redacted] | 1 | <e,e,e> | F23 | (4*e, 4*e, 4*e), (90, 90, 90) | 1 | 1 |
| 121 | T | [redacted] | 1 | <0,0,0> | T | [redacted] | 1 | <e,0,0> | F23 | (2*e, 2*e, 2*e), (90, 90, 90) | 1 | 1 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 122 | T | [redacted] | 1 | <e,e,e> | O | [redacted] | 1 | <0,0,0> | F432 | (4*e, 4*e, 4*e), (90, 90, 90) | 1 | 1 |
| 123 | O | [redacted] | 1 | <0,0,0> | O | [redacted] | 1 | <e,e,e> | P432 | (2*e, 2*e, 2*e), (90, 90, 90) | 1 | 1 |
| 124 | O | [redacted] | 1 | <0,0,0> | O | [redacted] | 1 | <e,0,0> | F432 | (2*e, 2*e, 2*e), (90, 90, 90) | 1 | 1 |

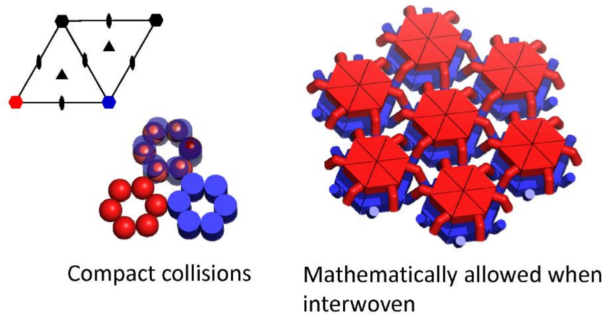
* Parenthetical a and b values in the Results column are used to disambiguate rare cases where the component symmetries and coordination numbers do not uniquely identify an SCM.
The table is available in readable csv format as Dataset 1.

**Supplementary Figures**

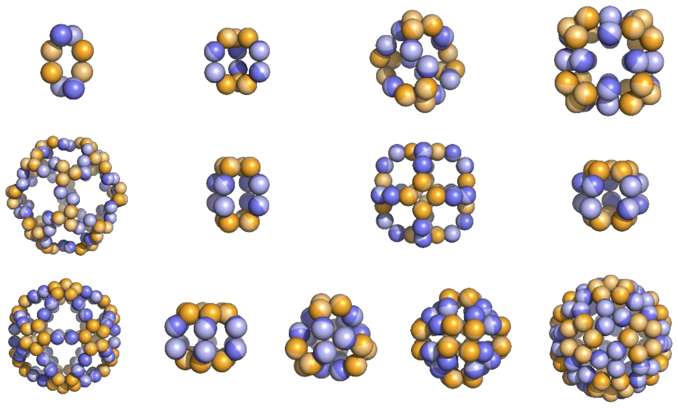
**A)** C6:{C2}{C3}



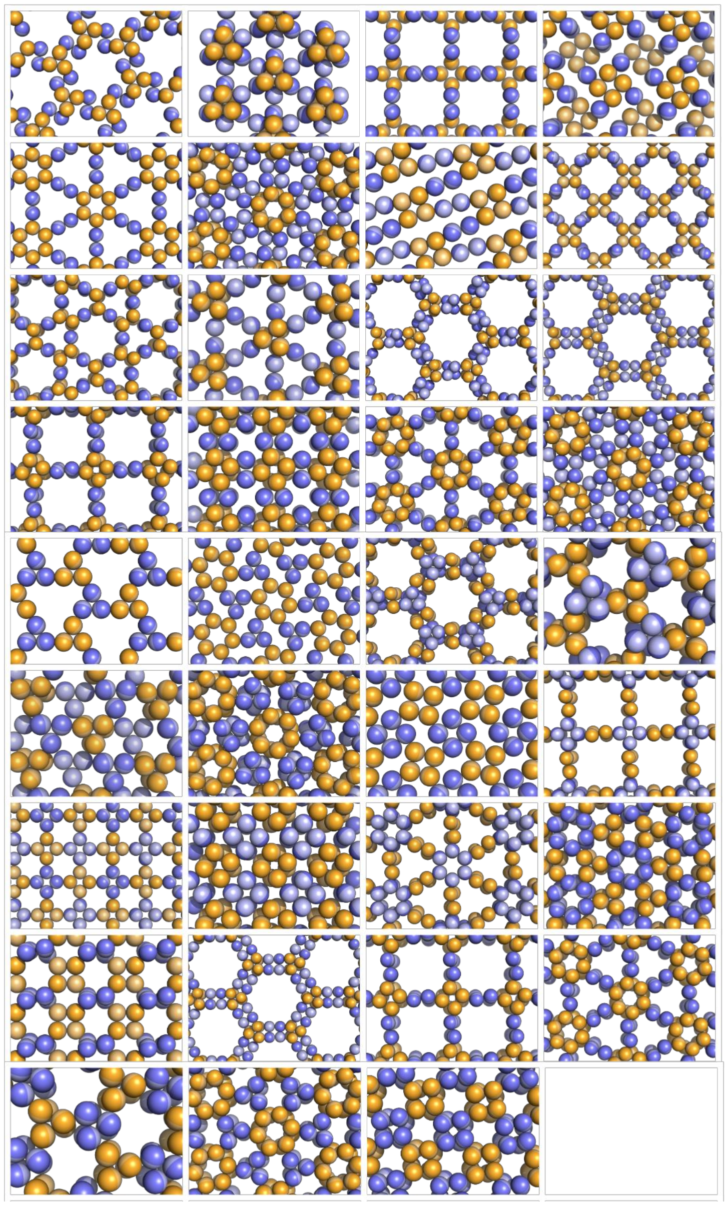
**B)** p6:{C6}{C6}



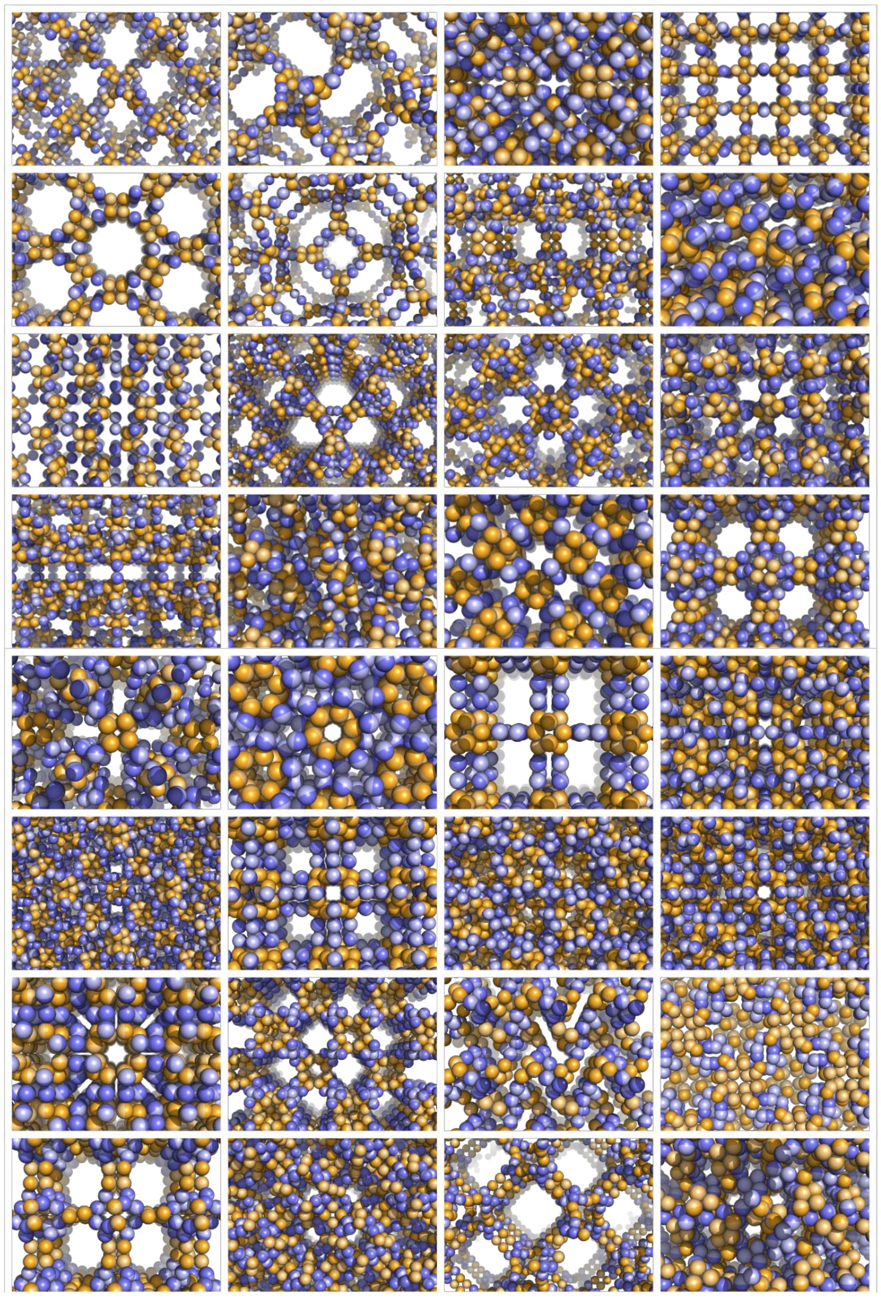
**Supplementary Figure S1.** Two examples (of essentially countless possibilities) where symmetry combinations are disallowed for compact shapes but mathematically allowed for elongated, interwoven shapes. (A) C2 combined with C3 with coincident axes to give C6, C6:{C2}{C3}. (top) Compact shapes are incompatible with this symmetry combination; collisions ensue. (bottom) Interweaving shows that the combination is mathematically allowable. (B) C6 combined with C6 with parallel axes to give layer symmetry p6, p6:{C6}{C6}. Compact hexamers of the two types collide immediately. Interweaving shows that the combination is mathematically allowable (12).



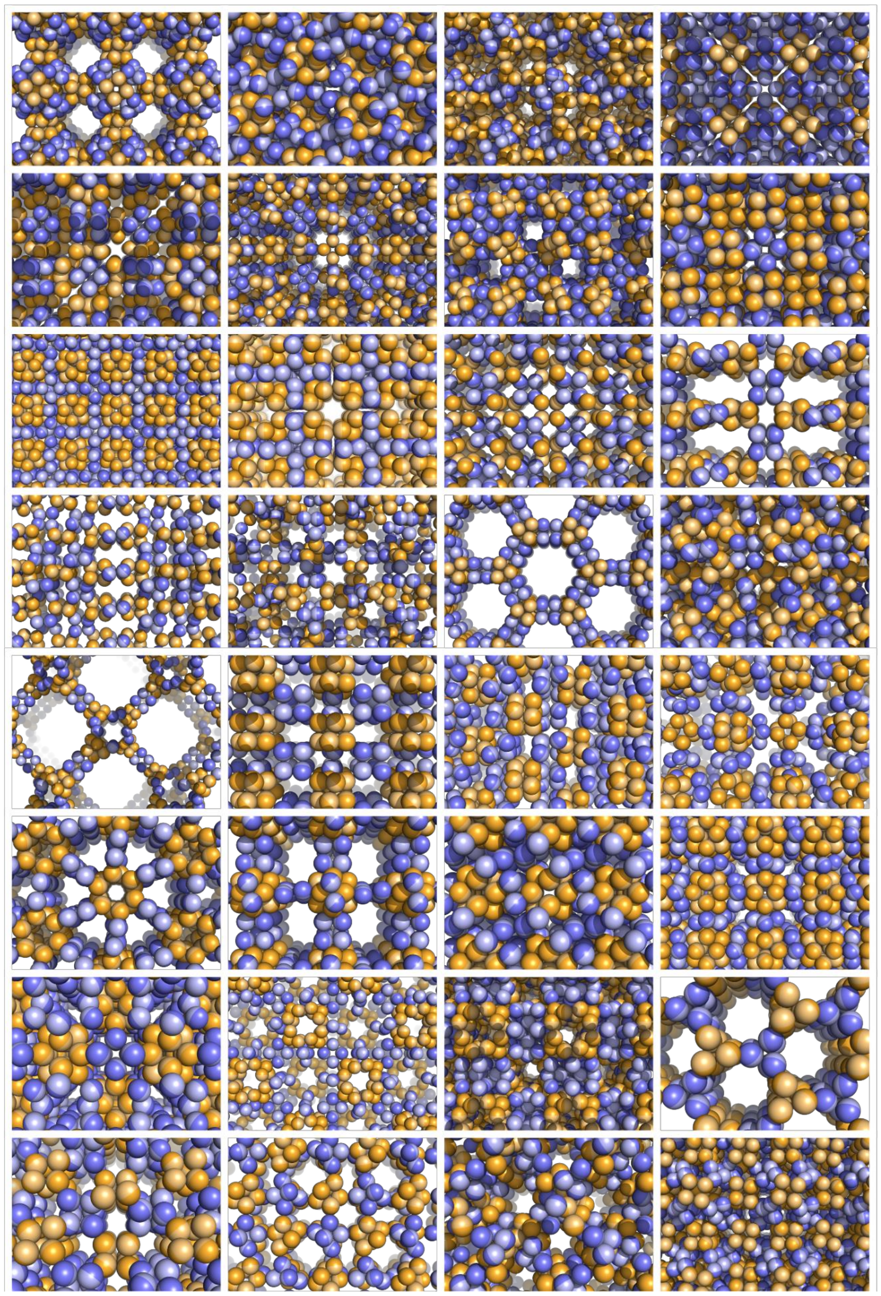
**Supplementary Figure S2.** Illustration and demonstration of constructability of the class of finite symmetry materials possible by combinations of two point symmetries. The corresponding symmetry combination types, indexed by their appearance in the master SCM table, are as follows:
D2:{C2}{C2}; D3:{C2}{C3}; T:{C2}{C3}; O:{C2}{C3};
I:{C2}{C3}; D4:{C2}{C4}; O:{C2}{C4}; D5:{C2}{C5};
I:{C2}{C5}; D6:{C2}{C6}; T:{C3}{C3}; O:{C3}{C4}; I:{C3}{C5}
An infinite set of dihedral types are of course possible but are not included here. The set shown is restricted to components useful for building higher symmetries, i.e. cubic, icosahedral, or extended materials in two or three dimensions. Dn with n > 6 is therefore excluded. Files for 3D visualization are provided at (https://people.mbi.ucla.edu/yeates/SCM_files/)

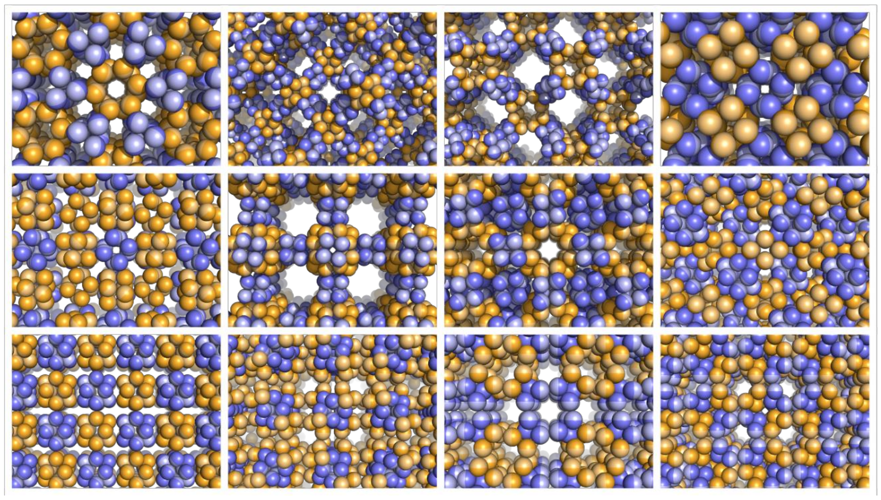


**Supplementary Figure S3.** Illustration and demonstration of constructability of the class of 2D layer symmetries possible by combinations of two point symmetries. The corresponding symmetry combination types, indexed by their appearance in the master SCM table, are as follows:
p6:{C2}{C3}; p312:{C2}{C3}; p4:{C2}{C4}; p4212:{C2}{C4};
p6:{C2}{C6}; p622:{C2}{C6}; c222:{C2}{D2}; p422:{C2}{D2};
p622:{C2}{D2}; p312:{C2}{D3}; p622:{C2}{D3}; p622:{C2}{D3};
p422:{C2}{D4}; p422:{C2}{D4}; p622:{C2}{D6}; p622:{C2}{D6};
p3:{C3}{C3}; p6:{C3}{C6}; p622:{C3}{D2}; p312:{C3}{D3};
p321:{C3}{D3}; p622:{C3}{D6}; p4:{C4}{C4}; p422:{C4}{D2};
p4212:{C4}{D2}; p422:{C4}{D4}; p622:{C6}{D2}; p622:{C6}{D3};
p222:{D2}{D2}; p622:{D2}{D3}; p422:{D2}{D4}; p622:{D2}{D6};
p312:{D3}{D3}; p622:{D3}{D6}; p422:{D4}{D4}
Files for 3D visualization are provided at (https://people.mbi.ucla.edu/yeates/SCM_files/)

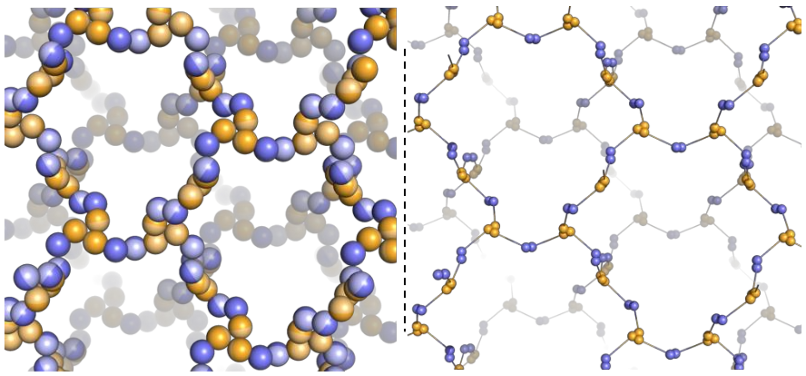


52





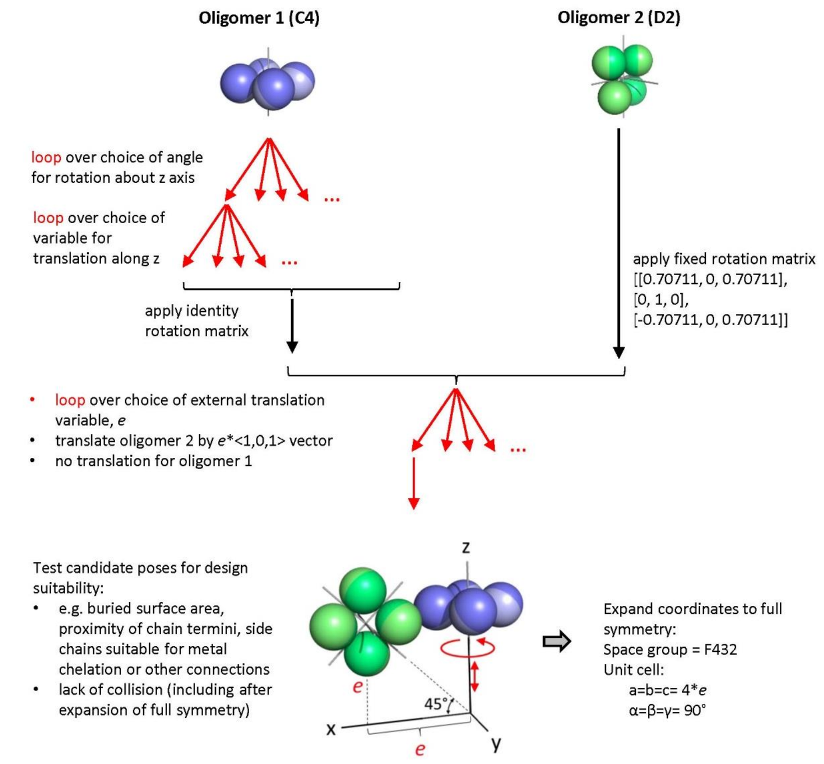
**Supplementary Figure S4.** Illustration and demonstration of constructability of the class of 3D crystal symmetries possible by combinations of two point symmetries. The corresponding symmetry combination types, indexed by their appearance in the master SCM table, are as follows:
I213:{C2}{C3}; P4132:{C2}{C3}; I432:{C2}{C4}; I4122:{C2}{D2};
P6222:{C2}{D2}; I432:{C2}{D2}; I4132:{C2}{D2}; R32:{C2}{D3};
P6322:{C2}{D3}; F4132:{C2}{D3}; I4132:{C2}{D3}; I432:{C2}{D3};
I4132:{C2}{D3}; P4132:{C2}{D3}; I422:{C2}{D4}; P432:{C2}{D4};
I432:{C2}{D4}; P622:{C2}{D6}; P23:{C2}{T}; F23:{C2}{T};
F4132:{C2}{T}; P432:{C2}{O}; F432:{C2}{O}; F432:{C2}{O};
P432:{C2}{O}; I432:{C2}{O}; P213:{C3}{C3}; F432:{C3}{C4};
P23:{C3}{D2}; F432:{C3}{D2}; I4132:{C3}{D2}; P4132:{C3}{D3};
P432:{C3}{D4}; F23:{C3}{T}; F432:{C3}{O}; P432:{C4}{C4};
I432:{C4}{D2}; F432:{C4}{D2}; I432:{C4}{D3}; P432:{C4}{D4};
F432:{C4}{T}; P432:{C4}{O}; F222:{D2}{D2}; P4222:{D2}{D2};
P6222:{D2}{D2}; P4232:{D2}{D2}; P622:{D2}{D3}; P4232:{D2}{D3};
I4132:{D2}{D3}; P422:{D2}{D4}; I422:{D2}{D4}; I432:{D2}{D4};
P622:{D2}{D6}; P23:{D2}{T}; P23:{D2}{T}; F432:{D2}{T};
P4232:{D2}{T}; F432:{D2}{O}; I432:{D2}{O}; P312:{D3}{D3};
P6322:{D3}{D3}; P4232:{D3}{D3}; P4132:{D3}{D3}; I432:{D3}{D4};
P622:{D3}{D6}; F4132:{D3}{T}; I432:{D3}{O}; P422:{D4}{D4};
P432:{D4}{D4}; P432:{D4}{O}; P432:{D4}{O}; F23:{T}{T};
F23:{T}{T}; F432:{T}{O}; P432:{O}{O}; F432:{O}{O}
Files for 3D visualization are provided at (https://people.mbi.ucla.edu/yeates/SCM_files/)



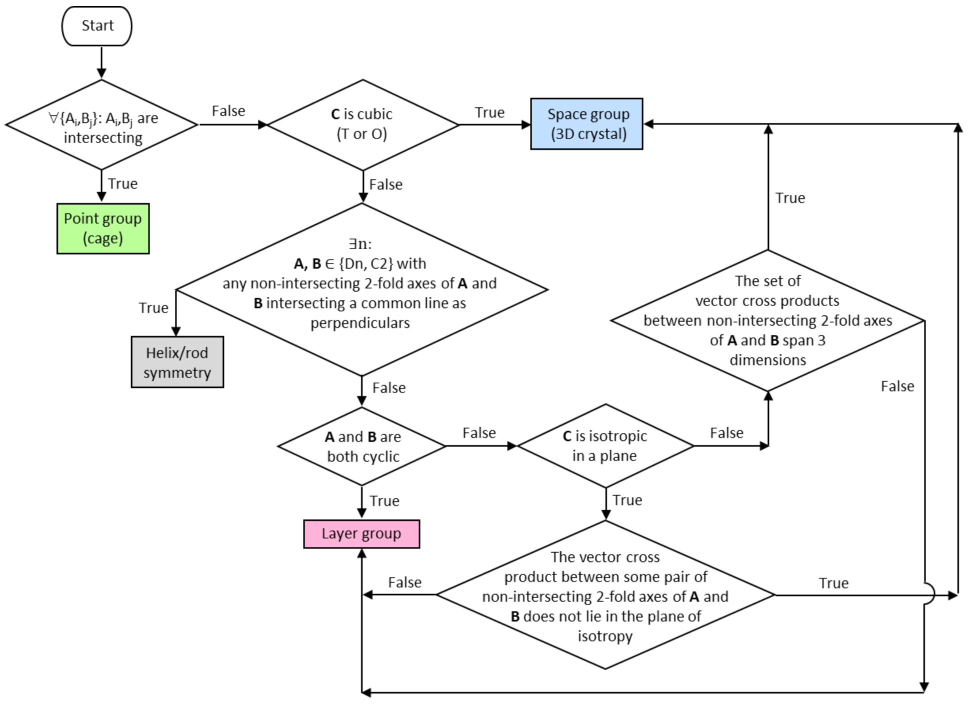
**Supplementary Figure S5**. Example of a design with a large ring size. I213:{C2}x{C3}, marble view (left), network view (right) has a ring size of 10.

**F432:{C4}{D2}**

| | Oligomer 1 {C4} | Oligomer 2 {D2} |
|---|---|---|
| **Internal rotations and translations** | Rotation and translation about z axis | None |
| **Orientation setting** | Identity | Rotation of z to the x-z diagonal [[√2/2,0, √2/2],[0,1,0],[-√2/2,0, √2/2]] |
| **External translations** | None | <*e*,0,*e*> or *e**<1,0,1> |


Oligomer 1 (C4)
Oligomer 2 (D2)
loop over choice of angle for rotation about z axis
...
loop over choice of variable for translation along z
...
apply identity rotation matrix
apply fixed rotation matrix [[0.70711, 0, 0.70711], [0, 1, 0], [-0.70711, 0, 0.70711]]
• loop over choice of external translation variable, e
• translate oligomer 2 by e*<1,0,1> vector
• no translation for oligomer 1
...
Test candidate poses for design suitability:
• e.g. buried surface area, proximity of chain termini, side chains suitable for metal chelation or other connections
• lack of collision (including after expansion of full symmetry)
z
e
45°
x
e
y
Expand coordinates to full symmetry:
Space group = F432
Unit cell:
a=b=c= 4*e
α=β=γ= 90°

**Supplementary Figure S6**. Detailed construction protocol for example SCM F432:{C4}{D2} according to specifications in Table S3. Computational loops corresponding to search degrees of freedom (of which there are 3 in this case) are emphasized in red. This example case expands on the middle panel of Figure 2 (main text). Each of the 124 SCM entries in Table S3 can be interpreted in similar fashion.


Start
∀{A_i,B_j}: A_i,B_j are intersecting
True
Point group (cage)
False
C is cubic (T or O)
True
Space group (3D crystal)
False
∃n: A, B ∈ {Dn, C2} with any non-intersecting 2-fold axes of A and B intersecting a common line as perpendiculars
True
Helix/rod symmetry
False
A and B are both cyclic
True
Layer group
False
C is isotropic in a plane
False
True
The set of vector cross products between non-intersecting 2-fold axes of A and B span 3 dimensions
True
False
The vector cross product between some pair of non-intersecting 2-fold axes of A and B does not lie in the plane of isotropy
False
True

**Supplementary Figure S7**. Decision flowchart showing how the relative arrangement of symmetry elements of the component symmetry groups of the two oligomers dictates the spatial dimensionality of the resulting SCM materials. $A_i$ and $B_j$ denote symmetry elements of the component point group symmetries **A** and **B**. **C** denotes the point group symmetry generated by the point groups underlying **A** and **B**. A symmetry is isotropic in the plane iff there is an axis of symmetry of higher order than 2 perpendicular to the plane. Outcomes are colored to match the multiplication table (Table 1 main text) for SCMs.

**Dataset S1**. Data table for construction parameterization of all SCMs (csv file).

**Dataset S2.** Pseudocode for all SCM constructions (text file).

## References

1. J. E. Padilla, C. Colovos, T. O. Yeates, Nanohedra: Using symmetry to design self assembling protein cages, layers, crystals, and filaments. *Proc. Natl. Acad. Sci.* **98**, 2217–2221 (2001).
2. J. C. Sinclair, K. M. Davies, C. Vénien-Bryan, M. E. M. Noble, Generation of protein lattices by fusing proteins with matching rotational symmetry. *Nat. Nanotechnol.* **6**, 558–562 (2011).
3. H. Garcia-Seisdedos, C. Empereur-Mot, N. Elad, E. D. Levy, Proteins evolve on the edge of supramolecular self-assembly. *Nature* **548**, 244–247 (2017).
4. H. Shen, *et al.*, De novo design of self-assembling helical protein filaments. *Science* **362**, 705–709 (2018).
5. E. H. Egelman, *et al.*, Structural Plasticity of Helical Nanotubes Based on Coiled-Coil Assemblies. *Structure* **23**, 280–289 (2015).
6. M. J. Pandya, *et al.*, Sticky-End Assembly of a Designed Peptide Fiber Provides Insight into Protein Fibrillogenesis. *Biochemistry* **39**, 8728–8734 (2000).
7. N. L. Ogihara, *et al.*, Design of three-dimensional domain-swapped dimers and fibrous oligomers. *Proc. Natl. Acad. Sci.* **98**, 1404–1409 (2001).
8. C. J. Tsai, R. Nussinov, A unified convention for biological assemblies with helical symmetry. *Acta Crystallogr. D Biol. Crystallogr.* **67**, 716–728 (2011).
9. K. A. Cannon, V. N. Nguyen, C. Morgan, T. O. Yeates, Design and Characterization of an Icosahedral Protein Cage Formed by a Double-Fusion Protein Containing Three Distinct Symmetry Elements. *ACS Synth. Biol.* **9**, 517–524 (2020).
10. Dey, S., Ritchie, D. & Levy, E. PDB-wide identification of biological assemblies from conserved quaternary structure geometry. *Nat Methods* **15,** 67–72 (2018). https://doi.org/10.1038/nmeth.4510
11. M. O'Keeffe, M. A. Peskov, S. J. Ramsden, O. M. Yaghi, The Reticular Chemistry Structure Resource (RCSR) Database of, and Symbols for, Crystal Nets. *Acc. Chem. Res.* **41**, 1782–1789 (2008).
12. T. O. Yeates, Geometric Principles for Designing Highly Symmetric Self-Assembling Protein Nanomaterials. *Annu. Rev. Biophys.* **46**, 23–42 (2017).

# CHAPTER 4

# A fragment-based protein interface docking algorithm for symmetric assemblies

# A Fragment-Based Protein Interface Design Algorithm for Symmetric Assemblies

Joshua Laniado, Kyle Meador and Todd O. Yeates

UCLA Department of Chemistry and Biochemistry
UCLA DOE Institute for Genomics and Proteomics
UCLA Molecular Biology Institute

Keywords:
protein design; symmetry; docking; self-assembly; secondary structure; protein interfaces; algorithms

## INTRODUCTION

A range of emerging bionanotechnology applications rely on designing protein molecules to bind and associate with each other in a geometrically specific fashion. Among such applications, those aimed at creating novel, self-assembling symmetric architectures, such as protein cages and extended protein arrays, place especially strict demands on achieving atomically precise associations. When such precision can be achieved by design, diverse protein-based materials with tailored spatial and biochemical properties can be produced. As examples, cubic and icosahedral protein cages [1–6], as well as extended protein arrays [7–10], are finding wide ranging uses as biotherapeutics (e.g. for vaccines) [11–13], as scaffolds for enzyme organization or atomic imaging [14–17], and as nanoscale containers for molecular encapsulation and delivery [18,19].

Owing to their complexity, as well as our incomplete understanding of their behavior, protein molecules present challenging subjects for design. In protein engineering studies, these challenges often manifest through unpredictable outcomes from mutagenesis, frequently leading to proteins that are prone to misfolding and aggregation. Improved computational methods are addressing those challenges, making it increasingly feasible to mutate the surface of two suitably chosen proteins to create a binding interface between them [20,21]. Similar goals are being reached using *de novo* polypeptides as components [22,23]. Yet, despite exciting progress, relatively low success rates are still common in application areas where precision and predictability are essential, generally requiring many design trials to achieve a smaller number of correctly assembling protein designs.

A general view on the current challenges in designing novel protein-protein interfaces is that computational methods do not necessarily generate (prospective) interfaces that mimic native protein-protein interfaces [24]. The difficulty of the task is heightened in design problems where additional spatial constraints must be met, beyond those required simply for binding. For the design of symmetric cages and regular arrays, for instance, the novel interface must bring the two component proteins together under exacting rules of symmetry; e.g., if each component is part of a naturally symmetric oligomer, then the interface must cause the symmetry axes of the separate components to intersect at a precisely prescribed angle. Such complex constraints confound the problem of designing optimal, native-like interfaces.

In addressing the problem of interface design in the context of symmetric assembly, the strategy introduced by King [2] prioritized the symmetric constraint part of the problem. There, oligomeric building blocks were docked by systematically sampling the rigid body degrees of freedom allowed by the point symmetry of the target assembly. As a result of the high dimensionality search space and the large number of different component oligomers considered for docking, a rapid first-pass scoring was used to identify configurations that were potentially suitable for design: the number of beta-carbon contacts between the docked oligomeric building blocks. Naturally, only a minute fraction of candidate poses chosen under such coarse criteria present interfaces that are similar in atomic detail to those from natural protein-protein complexes. Subsequent amino acid sequence design and additional filtering steps were required to identify interfaces that might exhibit native-like properties. Newer protocols have shown the value of considering known residue pair interactions during docking [25] and prioritizing interfacial hydrogen bonding during sequence design [5,26].

Recent exercises in protein design have begun to prioritize the consideration of secondary structure motifs and the atomic details of how they tend to associate in native proteins [27]. The expansive database of known protein structures provides valuable empirical frameworks for evaluating designed proteins in terms of secondary structure motifs. For instance, threading helical fragments together produces novel fold topologies that retain features of observed tertiary motifs [28,29]. Further, sequence design using statistical models of tertiary structure segments has competed with or outperformed physicochemical energy functions in routine design tasks [30]. The growing focus on secondary structure associations motivates an attempt to bring those principles to bear on the class of design problems related to symmetry-based assemblies.

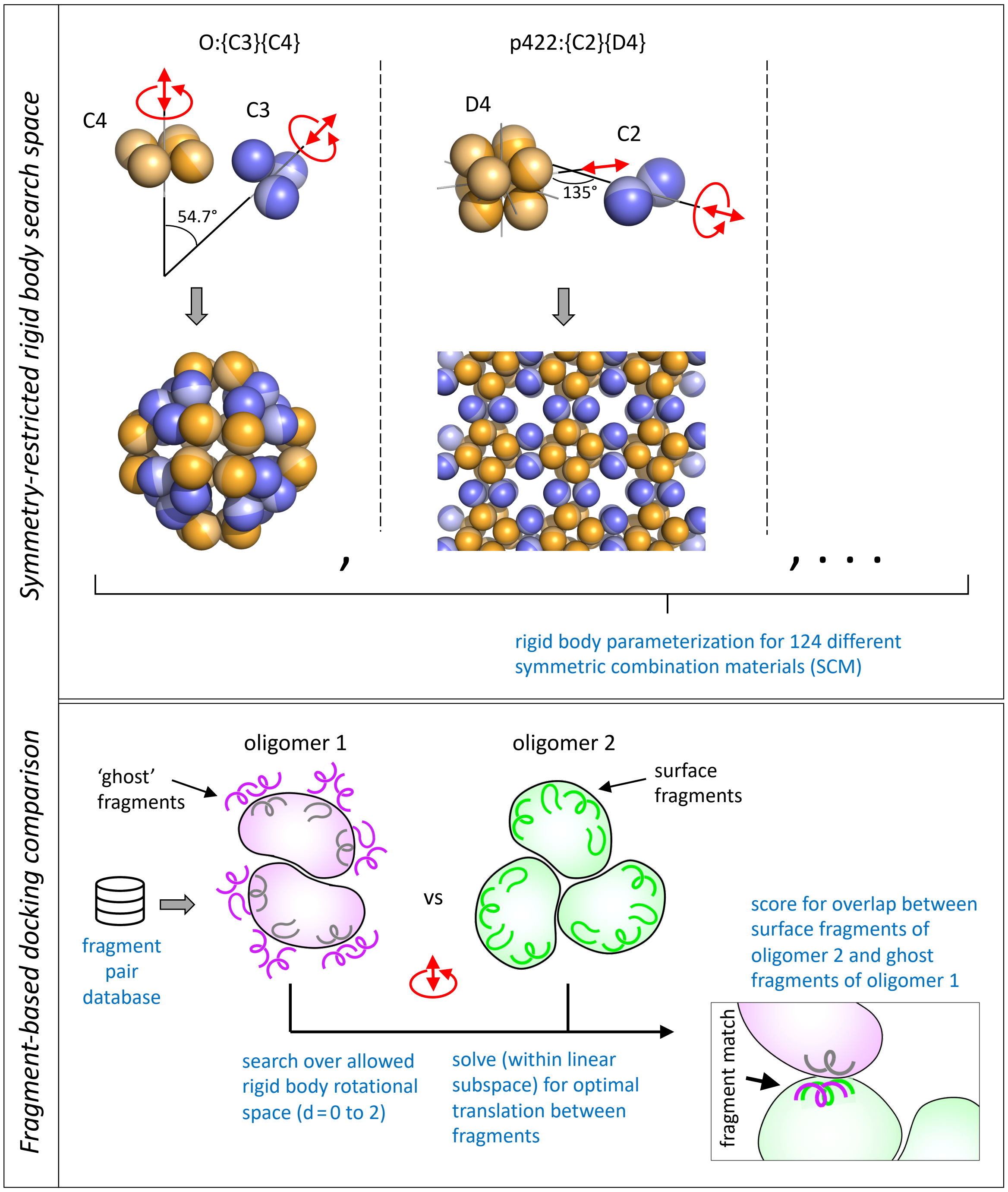
Here we describe algorithms and software that expand fragment-based design methodologies to symmetric docking applications – e.g. cubic cages and extended protein arrays. Our new program is parameterized to exploit recent theoretical work articulating the geometric rules for designing wide ranging nanoscale materials built from combinations of oligomeric protein components – i.e. symmetry combination materials (SCMs) [31]. Strategic choices are discussed for program optimization based on fragment-based lookup tables and separation of rotational vs translational subspace searches. Prospective novel designs are discussed, along with a retrospective analysis of successfully designed protein cages.

## RESULTS

### Docking under symmetry constraints

The goal of the program developed here was to enable fragment-based docking for the design of self-assembling materials based on the principles of combined symmetries. The essential idea for building highly symmetric materials from simpler protein oligomers was described by Padilla et al. [1], with diverse variations demonstrated in recent years [3,4,6,32]. A complete articulation of all possible SCMs was recently completed [31]. In addition to various cage types based on the Platonic solids, 35 kinds of 2-D arrays and 76 kinds of 3-D arrays were identified as targets possible for design. Each of the 124 SCMs presents a different set of rigid body constraints, and complementary rigid body degrees of freedom, for sampling allowable arrangements of the two oligomers to be docked. For the present work, we have integrated the design rules for all possible SCMs within a new program, Nanohedra.

We developed a general docking framework applicable to all SCMs that performs a search over multiple rigid body degrees of freedom relating two oligomeric building blocks (Fig. 1). The number of degrees of freedom depends on the symmetric system being constructed, ranging from a minimum of 1 to a maximum of 5 [31]. Exploiting advantages of pre-calculation methods, we were able to factor the search problem for all scenarios into a search over rotational degrees of freedom (for cases where they exist), followed by direct calculation of optimal translational values by linear algebra methods, thereby avoiding the need to explicitly search translational degrees of freedom for each rotation. Identifying favorable docking arrangements within the allowable rigid body search space is made possible by precomputing common protein-protein fragment configurations from known structural data.



Symmetry-restricted rigid body search space
O:{C3}{C4}
C4
C3
54.7°
p422:{C2}{D4}
D4
C2
135°
, , . . .
rigid body parameterization for 124 different symmetric combination materials (SCM)
Fragment-based docking comparison
oligomer 1
oligomer 2
'ghost' fragments
surface fragments
fragment pair database
vs
score for overlap between surface fragments of oligomer 2 and ghost fragments of oligomer 1
search over allowed rigid body rotational space (d = 0 to 2)
solve (within linear subspace) for optimal translation between fragments
fragment match

**Fig. 1** Scheme illustrating two major aspects of the Nanohedra program for designing symmetry combination materials (SCMs) from two oligomeric protein components. The top panel shows examples of two SCM types (of 124 types possible), focusing on the geometric rules that must be satisfied when bringing the two different oligomeric components into specific contact. In each case, the red arrows indicate the rigid body degrees of freedom available, which must be explored computationally in a search for favorable docking configurations that would be amenable to amino acid sequence design at the emergent protein-protein interface. Nanohedra encodes the specific rigid body parameterization required for constructing all 124 SCM types [31]. The bottom panel highlights the use of protein fragment pair libraries as the essential feature for selecting favorable design poses for subsequent interface design. This allows Nanohedra to generate native-like interfacial backbone arrangements for design. Program operation is

made computationally tractable through various pre-calculation schemes. One of these involves the decoration of the first oligomer with ‘ghost fragments’ (based on a library of favorable fragment pair configurations), after which the search for suitable docking poses is reduced to a problem of identifying allowable oligomeric arrangements wherein surface fragments belonging to oligomer 2 overlap closely with ghost fragments covering oligomer 1.

## Fragment-based elements

Focusing on short segments of protein structure makes it possible to reduce computational complexity with lookup or ‘hash’ tables. To this end, we chose to categorize local protein structure using five residue fragments. Heuristically, a segment of five residues is long enough to capture secondary structures types, including alpha helical and beta strand conformations, as well as loop structures, while being short enough to capture the allowable space of conformations with acceptable precision and coverage using a tractable number of representatives. Using a curated set of known protein-protein interfaces (see Methods), we computed the most highly represented fragment types found at interfaces using nearest neighbor clustering of carbon-alpha rmsd of each candidate fragment. We experimented with different similarity cutoff criteria, and settled on a 0.75 Å cluster inclusion limit which maximized fragment coverage, while ensuring stringent constraints on backbone geometry.

As expected, different fragment clusters were populated to different degrees; those representing canonical alpha helical and beta strand conformations were much more densely populated than those representing different loop conformations, or cases where regular secondary structure transitioned into loops. The top five clusters were sufficient to represent 61.4% of the candidate fragments; with the highest observed cluster corresponding to an alpha helical conformation, followed by a beta strand conformation then three loop or turn conformations. Rather than considering a larger number of individual fragment conformations, we retained a relatively small number at this stage of the analysis in order to maximize statistical power in subsequent steps describing diverse 3-D spatial associations between pairs of the fragment cluster types.

Following individual fragment clustering, a paired fragment-fragment clustering procedure was applied to inter-protein contacts from the same protein-protein interface set. This problem was simplified by a form of coordinate reduction. A set of three 'guide coordinates', built on the C-alpha atom of the central residue of the 5-residue fragment (Fig. S1), was associated with the representative fragment from each individual fragment cluster (see Methods); note that three x,y,z coordinates (nine variables) are sufficient to specify six dimensional rigid body orientation and position in three-dimensional space. This provided a generalized scheme for next analyzing the relative spatial arrangement between fragment-fragment pairs. Briefly, for every instance where a 5-residue fragment (type *i*) from one protein was found in spatial contact with a 5-residue fragment (type *j*) from another protein, each fragment in the *i,j* pair is assigned to one of the 5 individual fragment types. This allows placement of the representatives' guide coordinates onto the coordinate frame of the observed fragment pair. The guide coordinate pairing was then transformed to put the *i* guide coordinate set in a canonical setting (i.e. at the origin with internal axes along principle directions). The resulting coordinates of the *j* guide coordinate set are then stored, providing a full representation of the relative spatial arrangement of that instance of an *i,j* fragment pair. The *j* guide coordinates sets were used as the basis for a final nearest neighbors clustering step where the resulting cluster index, *k*, represents the different spatial modes in which the specific *i,j* fragment pairs tend to organize (Fig. 2). Clustering at this pairwise stage was based on a relatively strict similarity criterion (1 Å guide coordinate rmsd). In that way we were able to establish separate conformational and amino acid preferences for relatively finely discriminated fragment-fragment arrangements. The resulting data structure is a triplet of (*i,j,k*) indices, each carrying a 9-dimensional coordinate point that captures a frequently observed spatial relationship *k,* between a specific *i,j* fragment pair type. In addition, owing to the cartesian nature of the embedding, a 9x9 covariance matrix (nearly though not strictly obeying rank 6) provides a quadratic description of the spatial variation for the given *i,j,k* fragment pair cluster. Observed central residue amino acid frequencies are also stored for each *i,j,k* cluster and can be used to guide subsequent sequence design steps. Ultimately, a total of 97,935 *i,j* fragment pairs from observed structures were grouped into 4,530 *i,j,k* clusters specifying geometrically defined 3-D fragment associations.

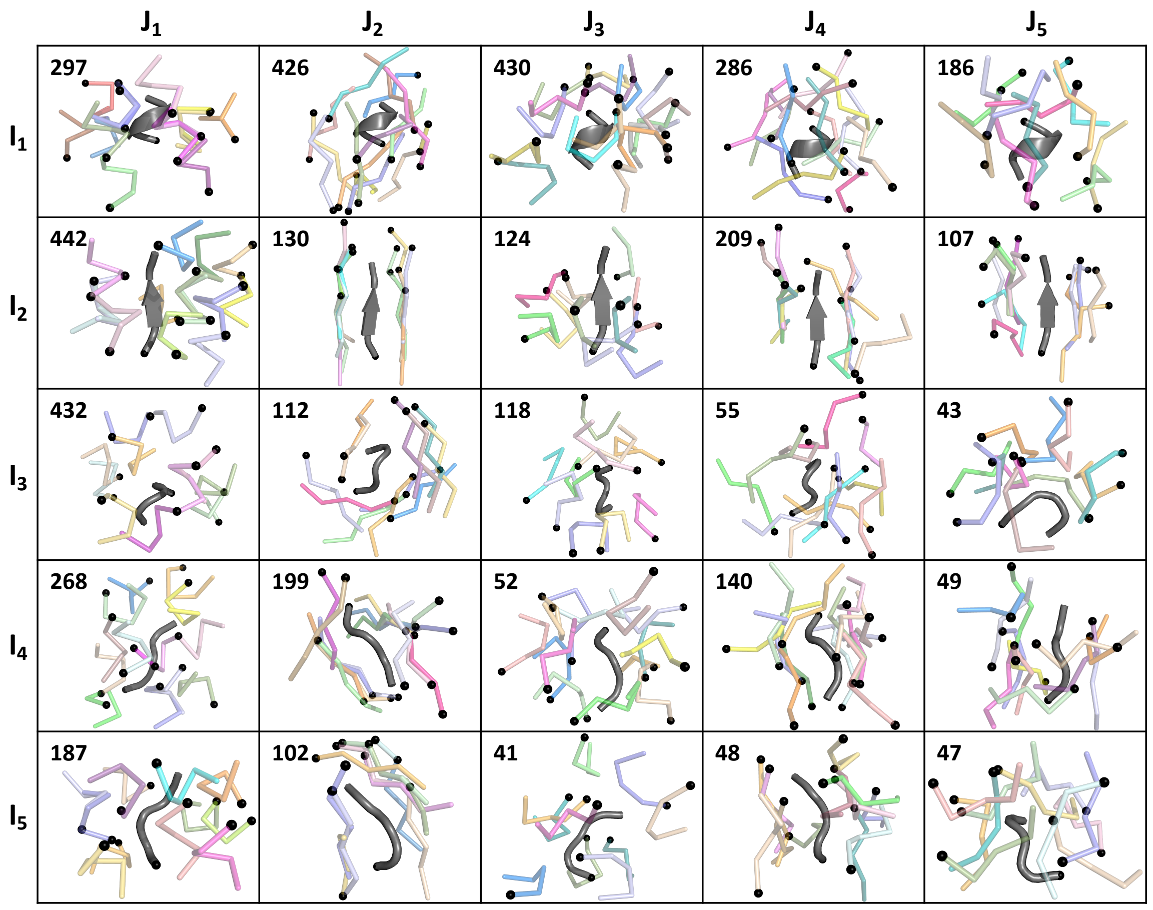


**Fig. 2** Interface fragment database. For each of the 25 *i,j* fragment pair possibilities, a single representative *i* fragment is shown in grey and a subset of cluster representatives of the top 20 most populated clusters are shown in color for the spatially clustered *j* fragments. N-termini are marked with black spheres. The total number of unique *i,j* clusters is indicated in the top left corner of each frame.

## Precoating by 'ghost fragments'

Having established common fragment pairing conformations in advance enables a pre-calculation protocol with important computational time savings. As a set-up to docking trials between two oligomers, one of the component oligomers (the first oligomer) is decorated with a large set of prospective 'ghost fragments' (Fig. S2 and Methods). These ghost fragments represent preferred interaction potentials based on the orientation of the fragments on the surface of the first oligomer. The precalculated database of representative *i,j,k* fragment pairs described previously serves as the source for constructing this set of ghost fragments. This ghost fragment set is intended to be inclusive for essentially all backbone configurations of the second oligomer that might comprise frequently observed interactions with the first. Depending on the size of the first protein, the ghost fragments may number in the thousands. Once the ghost fragment set is calculated, the subsequent fragment-based docking scheme is reduced to a problem of identifying orientations which might bring surface fragments of the second oligomer into near coincidence with the ghost fragments decorating the first.

**Solving for translation**

The outer loop of the docking calculations applies candidate rotation values to the two oligomers, if those degrees of freedom exist. The symmetric construction schemes for SCMs provide a maximum of one degree of rotational freedom for each component oligomer (e.g. about the unique symmetry axis of a cyclic oligomer). For each choice of rotational values for the two oligomers, a calculation is performed to test which of the possible pairs of fragments (chosen from the surface fragments of 2 and the ghost fragments of 1) are in nearly equivalent orientations, as would be required for near-overlap under any choice of translation. This step involves a large number of possible fragment pairs to be considered, as well as somewhat complex numerical calculations for orientation comparisons. We found it critical to shorten this calculation with further pre-calculation methods and hash tables. We assign each fragment (based on its guide coordinates) to a set of three Euler angles describing its orientation, with the Euler angles discretized into 10° bins. With a triplet of orientation indices assigned to each fragment, we are able to look up in a precalculated 6-dimensional Boolean (true/false) table whether or not the sets of triplets assigned to the two fragments are within a prescribed angular discrepancy (with an accuracy of roughly 10°).

The steps described above rapidly identify pairs of fragments (a surface fragment of oligomer 2 and a ghost fragment surrounding oligomer 1) that could be nearly coincident under the chosen orientation values and *some* translational values between the oligomers. It is critical however that the translational relationships conform to those that are prescribed by the particular symmetry rules of the SCM being constructed. Some SCM types have 3 translational degrees of freedom while some have as few as 1. Importantly, our program encodes those translational restrictions for all SCM types, based on tables provided in Laniado and Yeates [31]. For every pair of candidate fragments that have compatible orientations, our program calculates the optimal translation for overlap, *within the allowable space of rigid body translations for the given SCM type*. We then store the translational parameters for cases where the rmsd for the optimal overlap is within a prescribed cutoff (e.g. 1 Å).

Ultimately, a suitable docking arrangement between the oligomers is one where multiple candidate fragment pairs could be brought into near coincidence for the same (or highly similar) choices of the rotational and translational parameters. For fastest performance we found it efficacious to perform the docking analysis in a rapid first pass over a smaller set of candidate fragment pairs (e.g. requiring at least one helix-helix association), followed by a subsequent pass wherein the translational values established in the first pass restricted consideration of subsequent fragment pairs.

We found the procedures described above critical for reducing the CPU times to levels that were compatible with docking large sets of candidate oligomer pairs. Other approaches could also be considered, though we emphasize that procedures that might appear beneficial for certain kinds of symmetric construction choices are sometimes problematic for other types of constructions, e.g. depending on the types and numbers of the rigid body degrees of freedom. The system we developed applies universally to all 124 SCM types.

**Heuristic scoring**

For each pose, a Nanohedra score is calculated based on the collection of favorable fragment pairs identified, with the goal of evaluating how well the docked interface is supported by the underlying fragment observations. To compute the Nanohedra score, for each instance where a favorable surface fragment and ghost fragment pair has been identified, a similarity score (z) is first calculated by dividing the rmsd obtained between the surface and ghost fragments by the mean rmsd for member fragments comprising the ghost fragment's cluster (precalculated during fragment database creation), with a low value of z indicating a close similarity. If the z-score is less than a prescribed threshold value (e.g. 2), the inverse of 1 plus the squared z-score is taken to give a match score, ranging from 0 to 1 with 1 indicating a perfect match. This match score for each fragment is propagated to each of the five residues comprising those fragments on oligomers 1 and 2. In this way, each residue in the protein-protein interface might inherit multiple component scores, since each residue might belong to overlapping fragments participating in favorable fragment-fragment pairs. For each such interfacial residue, its assigned match score(s) are first ranked in descending order and are then weighted by $1/2^{\text{rank} - 1}$ (rank > 0) for a final summation. This weighting scheme bounds the final score for each residue to a maximum of 2. The weighted match scores are then summed across interfacial residues to give the final Nanohedra score for the identified docking configuration.

**Program considerations**

Nanohedra is a command line tool. It can be operated in one of three modes: Query, Dock or Post-processing. The Docking mode executes the main procedures described in the present work. The user specifies the desired symmetry material outcome, i.e. the specification of the two component symmetries and their resulting assembly type. Directory paths are input to specify the file locations for the oligomeric protein structures to be tested. The output provides pdb files with candidate docked poses in various forms (asymmetric unit within the final symmetry, docked oligomers, and an expanded symmetry). Other information includes the final Nanohedra score and the spatial transformation matrices mapping the canonically oriented coordinates onto the candidate pose. In order to guide subsequent design of the resulting interface, sequence information is output in the form of amino acid frequencies based on amino acid composition information tabulated from known structures contributing to the fragment database (Fig. S3).

The computer time for execution depends critically on the size of the proteins (because larger proteins carry more surface fragments), the number of rotational degrees of freedom for sampling, and the rotational sampling interval. Times on a single CPU core (2.5 GHz) can range from 2 to 24 hours. Computer memory requirements also depend on the sizes of the proteins and the size of the symmetry group generated by the final assembly. Requirements range from roughly 8 to 25GB. The user can override various default settings, e.g. for angular sampling in rotational searching or for the minimum number of fragment-fragment pairs needed for a well-docked pose.

The program can also be operated from the command line in a Query mode. This is an informational mode that helps the user understand different options and certain symmetry aspects of the material to be designed: e.g., what kinds of resulting SCM materials can be constructed from a given combination of components, and conversely what component oligomer types would be needed to construct different SCMs according to various target criteria, such as the dimensionality of the resulting material (cage vs layer vs three-dimensional crystal), the underlying rotational symmetry, or specific geometric features (like network properties) of the material to be designed. This mode captures the full space of available design materials recently articulated [31]. A final Post-processing mode provides tools for ranking output candidate poses, with options to sort by different criteria, e.g. according to the final Nanohedra matching score or according to the numbers of fragment pairs identified in the match.

The program is implemented in Python with the exception of one routine that is written in Fortran (orient_oligomer). Python dependencies include biopython [33], numpy [34], and scikit-learn (to implement the BallTree method for pairwise distance calculations and clash tests) [35]. Nanohedra also uses the freeSASA program to calculate solvent accessible surface areas [36].

## Prospective SCMs

To demonstrate the universality of our fragment-based docking approach, we constructed an array of prospective SCMs with representatives from point, layer, and space group symmetries, composed of various component symmetries ranging from C3 to T.

Nanohedra was run with default parameters, and for each combination of symmetric oligomers a search of the rigid body degrees of freedom inherent in each system produced numerous viable candidates with varied orientations and positions and different interfacial secondary structure compositions. A representative high scoring structure for each of the different SCMs produced is displayed in Figure 3. The results demonstrate the viability of the described method at producing numerous candidate assemblies conforming to a selected symmetric material while appearing native-like with respect to interfacial backbone-backbone associations.

In each example, the resulting interface exhibits high structural complementarity between oligomeric surfaces as can be seen by the degree of overlap between the ghost fragments of the first oligomer and the matched surface fragments of the second oligomer. The interfaces vary in the extent of secondary structure involvement, with each oligomer contributing at least one continuous secondary structure element to the interface and ranging from 8 (F23:{C3}{T}) to 20 (p222:{D2}{D2}) unique fragment matches. Matched interface fragments are sometimes enhanced by 'fortuitous' contacts involving seemingly less frequent or unobserved secondary structure interaction motifs. Many of the docked configurations comprise extensive helical interactions, with interfaces containing anywhere from two to five helices (see F23:{C3}{T} and I432:{C4}{D3}, respectively). The contribution from beta-strands is also apparent as both T:{C3}{C3} and p222:{D2}{D2} designs have mixed secondary structure interaction motifs involving helices and strands, despite prioritizing helix-helix pairs in first-pass searching.

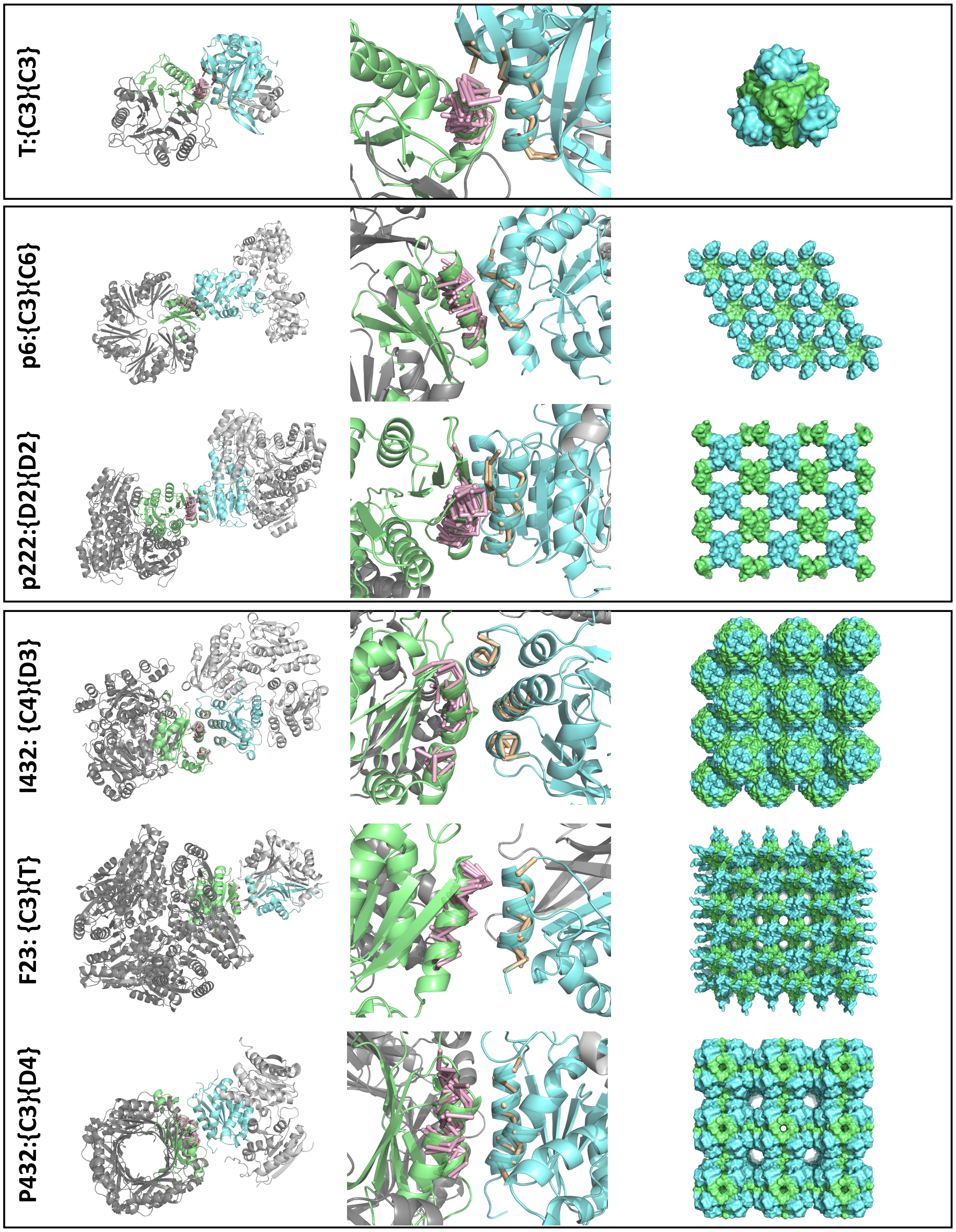


**Fig. 3** Prospective SCMs. Six example SCMs generated by Nanohedra are shown: a finite tetrahedral cage (top), two 2-D layers (middle) and three 3-D crystals (bottom). Column 1 illustrates the docked oligomeric building blocks that are required to construct the final material. Chains directly implicated in the docked interface are colored, while symmetrically related chains are in grey. Closeups of the interfaces are shown in column 2. Oligomer 1 (blue) surface fragments (tan) and associated ghost fragments (pink) are shown. Ghost fragments are matched with secondary

structure elements on the surface of oligomer 2 (green). The resulting symmetrically expanded materials are displayed in column 3. PDB accession codes used to generate the prospective materials shown are: 1OSC and 1NQ3 for T: {C3}{C3}, 4O5O and 1UAY for p222: {D2}{D2}, 1OSC and 1GTZ for F23: {C3}{T}, 2B34 and 3BBC for I432: {C4}{D3}, 1VHC and 2A10 for p6: {C3}{C6}, 4XCW and 1DHN for P432: {C3}{D4}.

## Post facto analysis of designed protein cages

The Nanohedra program provides a new tool for designing novel candidates for designed protein materials. Prior work in designing protein assemblies has shown the challenges of generating computational designs that produce the desired experimental outcomes; success rates remain relatively low, as discussed earlier. The favorable features of Nanohedra in allowing the selective construction of designs based on native-like interfaces will ultimately require experimental tests that are ongoing and not presented here. Nonetheless, the results of several recent design trials provide an opportunity to evaluate the prospective advantages of Nanohedra, ahead of new experimental trials.

For a retrospective analysis, we sought to demonstrate the efficacy of Nanohedra to prioritize computationally designed protein assemblies that went on to successful experimental validation in our earlier work, among the larger body of computational designs that led to experimental failure. We focused on designed protein cages, for which there are more than a dozen successful cases validated in atomic detail, along with hundreds of computational designs that led to experimental failure.

The available literature data provided us with a set of computational designs that had led to experimental success and another (much larger) set that had led to experimental failure. We ran Nanohedra on these designs to see if the two sets could be clearly distinguished; this would argue that Nanohedra has the capacity to generate computational designs that would have an improved experimental success rate. For each prior design (in both categories of experimental successes and failures), we took the two component oligomers in standard orientations (i.e. not corresponding to the previously designed configurations) and ran Nanohedra to generate prospective designs for symmetric cages of the desired symmetry. We then examined the ranked poses output by Nanohedra to see where in the list of candidate poses (if at all) we could find a configuration essentially matching the one that was generated and then experimentally tested in earlier work. For the group of experimental successes (of which there were 14), we were able to recapitulate 70% of the design targets within the top 12 scoring pose clusters and 100% of the targets within the top 88 ranked poses (Fig. 4). For the unsuccessful design set (of which there were 200), we repeated the same design procedure. Whereas for the successful design set we only had to go to rank order 12 to recapitulate 70% of the designs, for the set of experimental failures we had to go to rank 112 among output poses in order to recapitulate 70% of the designed poses. These calculations showed clearly that earlier computational designs that went on to experimental success are much more readily recapitulated in calculations using Nanohedra, compared to earlier computational designs that had failed. We further note that having to go to rank 12 or so to recapitulate most of the earlier experimental successes does preclude that poses ranked by Nanohedra above the ones designed earlier could quite plausibly have led to successful experimental constructions, different in orientation from the ones that were validated earlier.

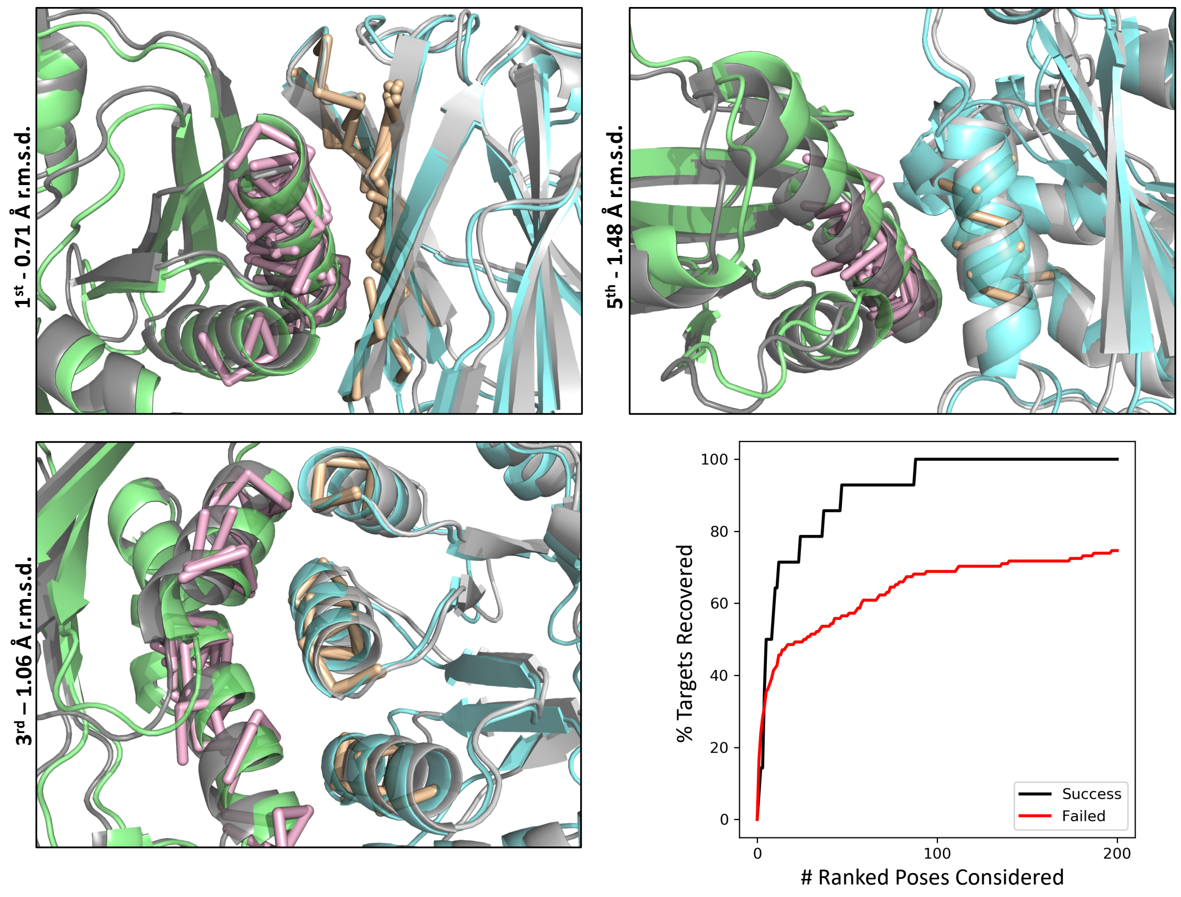


**Fig. 4** Post facto analysis of designed protein cages. Three representative examples of successfully recapitulated poses (top and bottom left). The crystal structures of the target designs are shown in grey and Nanohedra predictions are shown in color. The iRMSD indicates the agreement between the docked pose and the crystal structure. The numerical value indicates the rank of the docked pose. The crystal structure of T32-28 (4NWN) displays a 0.71 Å iRMSD with the first ranked pose (top left). The crystal structure of T33-15 (4NWO) exhibits a 1.48 Å iRMSD with the fifth ranked pose (top right). The crystal structure of I53-40 (5IM5) shows a 1.06 Å iRMSD with the 3rd ranked pose (bottom left). The ROC curve for all successful and failed designs shows the percentage of targets recovered according to the number of clustered Nanohedra poses considered (bottom right). iRMSD of 3.0 Å or less was considered as a recovered pose.

## DISCUSSION

Until now, designing symmetric protein assemblies has remained challenging to new entrants, as the process requires somewhat expert knowledge about symmetric construction and intertwined issues of how to sample allowable degrees of freedom in the context of docking software. Previously successful studies in executing two-component symmetric docking have used the Rosetta TCdock protocol, which requires the user to specify the symmetry rules through the use of symmetry definition files [3]. This is possible for symmetries already enumerated, but the majority of recently described SCM's [31] present a remaining challenge for specifying allowable degrees of freedom in the context of existing design software. By enumerating the allowable degrees of freedom for each SCM into a comprehensive, simple to use framework, Nanohedra will empower a broader group of users to explore the large space of possible symmetric designed materials. This should accelerate the development of novel designed materials by protein and biomaterial engineers.

Nanohedra harnesses the power of a recently established theoretical framework [31] to enable the construction of a universe of possible protein-based nanomaterials. Nanohedra's docking algorithm implements a novel fragment-based approach for assessing whether docked solutions resemble biological interfaces. Importantly, the Nanohedra score does not depend on simplified heuristics for rapid assessment of binding likelihood (e.g. number of Cbeta-Cbeta contacts), a notable difference from established docking methods [38–40]. Instead, its statistical representation of clustered secondary structure elements exploits empirical knowledge of typical packing motifs utilized in native protein interfaces. The implications of this choice, as demonstrated in our retrospective analysis of successful versus failed designs, are notable. In agreement with previous findings [37], geometric packing of secondary structure alone captures many essential elements of protein interaction. Furthermore, although Nanohedra is geared for design of symmetric materials, our results point to potential opportunities for advances in macromolecular docking and interface design in other contexts.

This first version of Nanohedra will admit future improvements along various lines, including GPU enhancements to increase speed. Critical experimental studies will be needed to evaluate the most important assertions concerning the expected advantages of generating assemblies with native-like interfaces. We also emphasize that successful design requires judicious amino acid interface design as a final step. This subsequent design step is separate from the construction of native-like (backbone-level) poses provided by Nanohedra, though as noted above Nanohedra provides valuable information about specific amino acid preferences favored in the fragment interfaces. This position specific frequency information can be exploited by sequence design programs such as Rosetta in the final step of design.

## METHODS

### Fragment database generation

To generate the fragment library, all non-redundant, biologically relevant interfaces from high resolution structures were gathered from the PDB. For homomers, QSBio [41] verified structure codes and biological assemblies were referenced to pull all assemblies with a confidence ranking of high or very high. For heteromers, biological assemblies were identified using PISA [42]. The homo- and heteromer sets were next filtered to include only representative structures clustered at 90% sequence identity with a reported resolution <= 2.0 Å, experimental expression in *E. coli*, no nucleic acids and no membrane proteins. For each identified structure, all unique interfaces between two chains were extracted with the exclusion of chains with less than 10 residues and if the interface contained fewer than 5 beta-carbon atoms of one chain within 8 Å of the second chain. From the resulting chain pairs, inter-chain beta-carbon distances were computed and residues that were 8 Å or less apart were selected as residue pairs across the interface. For each residue in the interface residue pair, the preceding and following two residues (i.e. i-2 through i+2) were included in the observation and the resulting five residue segments were stored, first as an individual residue segment (mono fragment), and second, as a pair of segments across the interface (fragment pairs). For residues with multiple conformations, the A conformation was chosen. Selenomethionine residues were not considered.

From the pool of individual fragments, a subset was chosen to perform all against all rmsd measurements followed by nearest neighbor clustering. The top five neighbor clusters were selected as the clustering population significantly decreased after this point. From each of the top five clusters, the fragment with the most neighbors was selected as a cluster representative, centered on the origin, and stored. As for the saved paired fragments, both fragments in the pair were classified for membership to one of the top five individual fragment representatives according to a CA RMSD of 0.75 Å. If one of the fragments in the pair was not represented by an individual fragment representative, the pair was discarded from further classification. Next, each fragment in the fragment pair was subjected to a structural superimposition on its corresponding matched fragment representative. This centered one fragment in the pair at the origin aligned to its structural representative, while maintaining the relative position of the partner fragment to this aligned fragment. Once in this orientation, a set of three coordinates was stored, one coordinate at the partner fragments central CA atom, and another two a unit vector away on the x or the y axis. This coordinate set, stored for each fragment observation, describes the transformation of the partner fragment's central CA atom, and its relative orientation in respect to the aligned fragment. In this way, each partner fragment marked a unique observation of the spatially encoded and secondary structure dependent interaction potential surrounding the individual fragment representatives.

Finally, for each individual fragment representative, and for each set of secondary structure dependent guide coordinates of that fragment representative, a subset of those guide coordinates was subjected to all against all rmsd calculations followed by nearest neighbor clustering. The resulting guide coordinate clusters were binned with a maximum of 1 Å translational and rotational deviation, requiring at least 4 members in the cluster to be considered. From this set of guide coordinate clusters, all possible guide coordinates were subjected to membership in the resulting clusters by testing for minimal rmsd. If an rmsd below 1 A could not be located, the guide coordinates were disregarded as outliers. Search proceeded for each partner secondary structure associated with each fragment representative.

For each i,j,k fragment pair cluster, the cluster size and the cluster representative fragment coordinates and guide coordinates are stored. Additionally, the mean guide coordinate rmsd and observed amino acid pair frequencies for central fragment residues are computed and then stored. The top 75% most populated i,j,k clusters were then chosen for our final fragment database.

## Docking prospective SCMs

For each SCM, homo-oligomers matching the design criteria were curated from the PDB by searching for the desired point group symmetry, X-ray resolution less than 2.5 Å, a helical content greater than 30% and *Escherichia coli* as the organism used for protein expression. Structures containing membrane proteins or nucleic acids were removed. Biological assemblies were identified using QSBio [41] and representatives clustered at 70% sequence identity were then downloaded from the Protein Data Bank. A few candidate oligomeric building blocks were then selected for pair-wise docking with Nanohedra using the default parameters.

**Design recapitulation**

The dataset for the design recapitulation experiments was generated by selecting all successfully designed (structural agreement with the model) two-component tetrahedral, octahedral and icosahedral designs from previously published work [3–5,11,13,43] as well as the failed designs (described as insoluble or unknown oligomerization state) from King, N. et al. 2014 and Bale J. et al. 2016 [3,4].

For each successful design, the two component oligomers used for docking were extracted from the deposited PDB structure of the protein cage. For the failed designs, the PDB structures of the native oligomeric building blocks were used. Default Nanohedra docking parameters were used with the exception of a 2° rotational sampling step instead of 3° for each component oligomer. Docking proceeded until all rotational degrees of freedom had been sampled. For 4NWN we had to modify the initial default Helix-Helix fragment search to Strand-Helix. Since the dimeric component is mainly composed of beta-strands on its surface, suitable docked configurations could not be identified with the default initial Helix-Helix search. Only the default Helix-Helix fragment search was used for failed designs and designs were not considered if they didn't have at least one helix-helix interaction (only a very few cases).

The carbon-alpha interface RMSD (iRMSD) was computed between the target design and each Nanohedra output pose. For successfully designed structures, the coordinates deposited in the PDB were used as a reference. Models of the failed designs were obtained from Neil King and Jacob Bale upon request. For each design target, the 2000 docked poses with the lowest iRMSD to the design target were selected and nearest neighbor clustering was performed using all to all iRMSD calculations. Interfaces within 1Å iRMSD threshold were clustered, then each cluster was ranked according to the Nanohedra score of the cluster representative.

## CODE AVAILABILITY

The Nanohedra source code is freely available at https://github.com/nanohedra/nanohedra

## ACKNOWLEDGEMENTS

This work was supported by NSF Grant: CHE-1629214. We thank Duilio Cascio and Alex Lisker for computing support and Michael Sawaya for helpful discussions. We thank Neil King and Jacob Bale for providing us with the models of their two-component cage designs.

## AUTHOR CONTRIBUTIONS

The research was conceived by TOY and JL. The code was written by TOY and JL. The fragment database was constructed by JL. The example SCMs were constructed by JL and KM. The post facto analysis of designed protein cages was performed by KM and JL. The written manuscript was prepared by TOY, KM and JL.

## NOTES

The authors declare no conflicting interests.

**Supplementary Information for:**

# A fragment-based protein interface design algorithm for symmetric assemblies

Joshua Laniado, Kyle Meador and Todd O. Yeates

**Contents:**

## SUPPLEMENTARY FIGURES

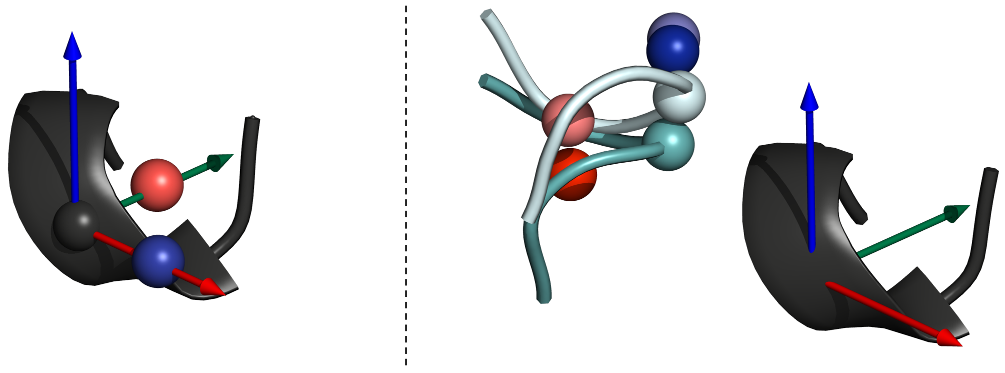


**Fig. S1.** Guide Coordinates

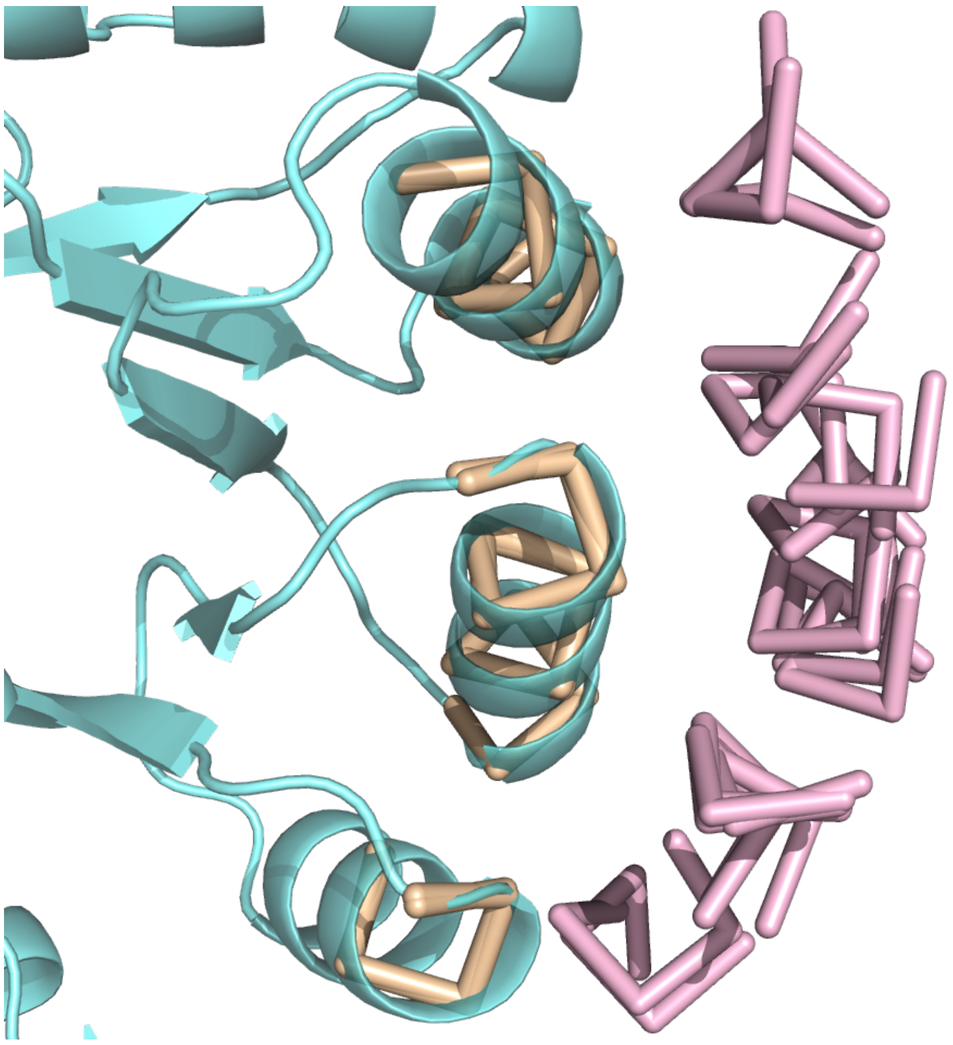


**Fig. S2.** Ghost fragments. Oligomer 1 (cyan), its surface fragments (tan) and associated ghost fragments (pink).

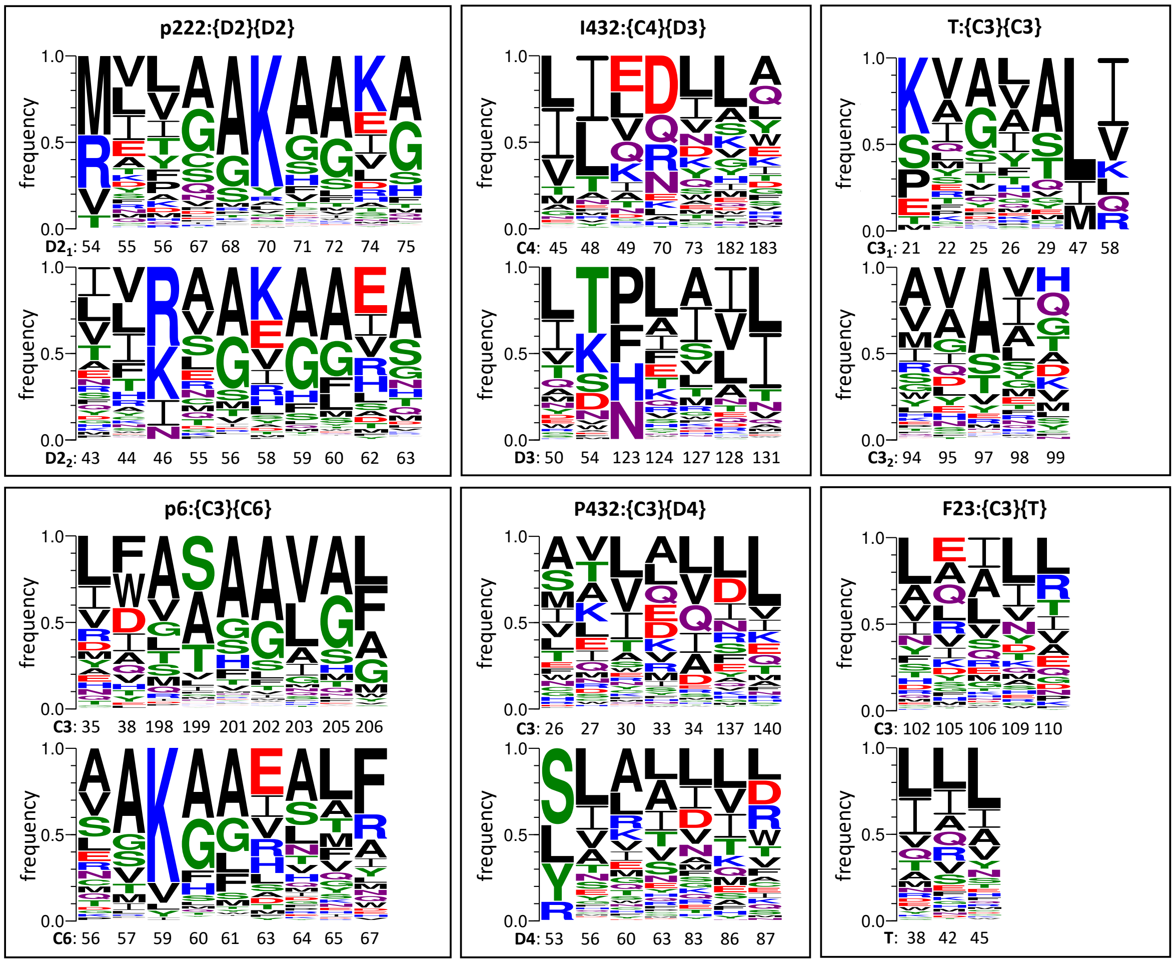


**Fig. S3.** Potential amino acid preferences for prospective SCMs in Figure 3

## SUPPLEMENTARY TEXT

PDB IDs and design names used for recapitulation experiments

PDB IDs of the experimentally validated designs:

4NWN, 4NWO, 4NWP, 4NWR, 4ZK7, 5CY5, 5IM4, 5IM5, 5IM6, 6P6F, 6VFH, 6VFI, 6VFJ, 6VL6

Names of the 'failed' designs:

I32-01, I32-03, I32-05, I32-07, I32-08, I32-12, I32-13, I32-14, I32-15, I32-16, I32-17, I32-20, I32-22, I32-23, I32-24, I32-25, I32-27, I32-31, I32-33, I32-34, I32-35, I32-36, I32-37, I32-38, I32-39, I32-40, I32-41, I32-45, I32-46, I32-49, I32-52, I32-53, I32-54, I32-55, I32-56, I32-60, I32-62, I32-

64, I32-65, I32-68, I32-70, I52-01, I52-07, I52-09, I52-10, I52-11, I52-12, I52-14, I52-17, I52-18, I52-20, I52-22, I52-23, I52-24, I52-26, I52-27, I52-28, I52-29, I52-31, I52-34, I52-35, I52-36, I52-38, I52-39, I52-40, I52-41, I52-42, I52-43, I52-44, I52-46, I53-06, I53-09, I53-12, I53-15, I53-16, I53-21, I53-23, I53-25, I53-27, I53-28, I53-29, I53-33, I53-35, I53-37, I53-43, I53-48, I53-49, I53-58, I53-61, I53-62, I53-63, I53-64, I53-67, I53-68, I53-70, I53-72, I53-74, I53-75, I53-79, I53-80, I53-82, I53-83, T32-02, T32-03, T32-06, T32-07, T32-08, T32-09, T32-11, T32-17, T32-18, T32-20, T32-24, T32-25, T32-26, T32-27, T33-01, T33-02, T33-03, T33-04, T33-05, T33-06, T33-07, T33-08, T33-11, T33-12, T33-13, T33-14, T33-16, T33-17, T33-18, T33-22, T33-23, T33-24, T33-25, T33-26, T33-27, T33-29

RPX designs were those utilizing the motif library from [4] which included I32 and I52 designs. All others are Non-RPX designs.

## REFERENCES

(1) Padilla, J. E.; Colovos, C.; Yeates, T. O. Nanohedra: Using Symmetry to Design Self Assembling Protein Cages, Layers, Crystals, and Filaments. *Proceedings of the National Academy of Sciences* 2001, *98* (5), 2217–2221. https://doi.org/10.1073/pnas.041614998.

(2) King, N. P.; Sheffler, W.; Sawaya, M. R.; Vollmar, B. S.; Sumida, J. P.; André, I.; Gonen, T.; Yeates, T. O.; Baker, D. Computational Design of Self-Assembling Protein Nanomaterials with Atomic Level Accuracy. Science 2012, *336* (6085), 1171–1174. https://doi.org/10.1126/science.1219364.

(3) King, N. P.; Bale, J. B.; Sheffler, W.; McNamara, D. E.; Gonen, S.; Gonen, T.; Yeates, T. O.; Baker, D. Accurate Design of Co-Assembling Multi-Component Protein Nanomaterials. *Nature* 2014, *510* (7503), nature13404. https://doi.org/10.1038/nature13404.

(4) Bale, J. B.; Gonen, S.; Liu, Y.; Sheffler, W.; Ellis, D.; Thomas, C.; Cascio, D.; Yeates, T. O.; Gonen, T.; King, N. P.; Baker, D. Accurate Design of Megadalton-Scale Two-Component Icosahedral Protein Complexes. Science 2016, *353* (6297), 389–394. https://doi.org/10.1126/science.aaf8818.

(5) Cannon, K. A.; Park, R. U.; Boyken, S. E.; Nattermann, U.; Yi, S.; Baker, D.; King, N. P.; Yeates, T. O. Design and Structure of Two New Protein Cages Illustrate Successes and Ongoing Challenges in Protein Engineering. *Protein Sci* 2020, *29* (4), 919–929. https://doi.org/10.1002/pro.3802.

(6) Cannon, K. A.; Nguyen, V. N.; Morgan, C.; Yeates, T. O. Design and Characterization of an Icosahedral Protein Cage Formed by a Double-Fusion Protein Containing Three Distinct Symmetry Elements. *Acs Synth Biol* 2020. https://doi.org/10.1021/acssynbio.9b00392.

(7) Sinclair, J. C.; Davies, K. M.; Vénien-Bryan, C.; Noble, M. E. Generation of Protein Lattices by Fusing Proteins with Matching Rotational Symmetry. *Nature Nanotechnology* 2011, *6* (9), 558. https://doi.org/10.1038/nnano.2011.122.

(8) Gonen, S.; DiMaio, F.; Gonen, T.; Baker, D. Design of Ordered Two-Dimensional Arrays Mediated by Noncovalent Protein-Protein Interfaces. Science 2015, *348* (6241), 1365–1368. https://doi.org/10.1126/science.aaa9897.

(9) Ben-Sasson, A. J.; Watson, J.; Sheffler, W.; Johnson, M. C.; Bittleston, A.; Somasundaram, L.;

Decarreau, J.; Jiao, F.; Chen, J.; Drabek, A. A.; Jarrett, S. M.; Kollman, J. M.; Blacklow, S. C.; Yoreo, J. J. D.; Ruohola-Baker, H.; Derivery, E.; Baker, D. Design of Biologically Active Binary Protein 2D Materials. https://doi.org/10.1101/2020.09.19.304253.

(10) Suzuki, Y.; Cardone, G.; Restrepo, D.; Zavattieri, P. D.; Baker, T. S.; Tezcan, F. A. Self-Assembly of Coherently Dynamic, Auxetic, Two-Dimensional Protein Crystals. Nature 2016, *533* (7603), 369. https://doi.org/10.1038/nature17633.

(11) Ueda, G.; Antanasijevic, A.; Fallas, J. A.; Sheffler, W.; Copps, J.; Ellis, D.; Hutchinson, G. B.; Moyer, A.; Yasmeen, A.; Tsybovsky, Y.; Park, Y.-J.; Bick, M. J.; Sankaran, B.; Gillespie, R. A.; Brouwer, P. J.; Zwart, P. H.; Veesler, D.; Kanekiyo, M.; Graham, B. S.; Sanders, R. W.; Moore, J. P.; Klasse, P. J.; Ward, A. B.; King, N. P.; Baker, D. Tailored Design of Protein Nanoparticle Scaffolds for Multivalent Presentation of Viral Glycoprotein Antigens. *eLife* 2020. https://doi.org/10.7554/eLife.57659.

(12) Marcandalli, J.; Fiala, B.; Ols, S.; Perotti, M.; Schueren, W. de van der; Snijder, J.; Hodge, E.; Benhaim, M.; Ravichandran, R.; Carter, L.; Sheffler, W.; Brunner, L.; Lawrenz, M.; Dubois, P.; Lanzavecchia, A.; Sallusto, F.; Lee, K. K.; Veesler, D.; Correnti, C. E.; Stewart, L. J.; Baker, D.; Loré, K.; Perez, L.; King, N. P. Induction of Potent Neutralizing Antibody Responses by a Designed Protein Nanoparticle Vaccine for Respiratory Syncytial Virus. *Cell* 2019, *176* (6), 1420-1431.e17. https://doi.org/10.1016/j.cell.2019.01.046.

(13) Brouwer, P. J. M.; Antanasijevic, A.; Berndsen, Z.; Yasmeen, A.; Fiala, B.; Bijl, T. P. L.; Bontjer, I.; Bale, J. B.; Sheffler, W.; Allen, J. D.; Schorcht, A.; Burger, J. A.; Camacho, M.; Ellis, D.; Cottrell, C. A.; Behrens, A.-J.; Catalano, M.; Moral-Sánchez, I. del; Ketas, T. J.; LaBranche, C.; Gils, M. J. van; Sliepen, K.; Stewart, L. J.; Crispin, M.; Montefiori, D. C.; Baker, D.; Moore, J. P.; Klasse, P. J.; Ward, A. B.; King, N. P.; Sanders, R. W. Enhancing and Shaping the Immunogenicity of Native-like HIV-1 Envelope Trimers with a Two-Component Protein Nanoparticle. *Nat Commun* 2019, *10* (1), 4272. https://doi.org/10.1038/s41467-019-12080-1.

(14) Heater, B. S.; Yang, Z.; Lee, M. M.; Chan, M. K. In Vivo Enzyme Entrapment in a Protein Crystal. *J Am Chem Soc* 2020. https://doi.org/10.1021/jacs.9b13462.

(15) Ernst, P.; Plückthun, A.; Mittl, P. R. E. Structural Analysis of Biological Targets by Host:Guest Crystal Lattice Engineering. *Sci Rep-uk* 201*9*, 9 (1), 15199. https://doi.org/10.1038/s41598-019-51017-y.

(16) McConnell, S. A.; Cannon, K. A.; Morgan, C.; McAllister, R.; Amer, B. R.; Clubb, R. T.; Yeates, T. O. Designed Protein Cages as Scaffolds for Building Multienzyme Materials. Acs Synth Biol

2020. https://doi.org/10.1021/acssynbio.9b00407.

(17) Liu, Y.; Huynh, D. T.; Yeates, T. O. A 3.8 Å Resolution Cryo-EM Structure of a Small Protein Bound to an Imaging Scaffold. Nat Commun 2019, 10 (1), 1864. https://doi.org/10.1038/s41467-019-09836-0.

(18) Liang, M.; Fan, K.; Zhou, M.; Duan, D.; Zheng, J.; Yang, D.; Feng, J.; Yan, X. H-Ferritin–Nanocaged Doxorubicin Nanoparticles Specifically Target and Kill Tumors with a Single-Dose Injection. *Proc National Acad Sci* 2014, *111* (41), 14900–14905. https://doi.org/10.1073/pnas.1407808111.

(19) Edwardson, T. G. W.; Tetter, S.; Hilvert, D. Two-Tier Supramolecular Encapsulation of Small Molecules in a Protein Cage. Nat Commun 2020, 11 (1), 5410. https://doi.org/10.1038/s41467-020-19112-1.

(20) Fleishman, S. J.; Whitehead, T. A.; Ekiert, D. C.; Dreyfus, C.; Corn, J. E.; Strauch, E.-M.; Wilson, I. A.; Baker, D. Computational Design of Proteins Targeting the Conserved Stem Region of Influenza Hemagglutinin. Science 2011, *332* (6031), 816–821. https://doi.org/10.1126/science.1202617.

(21) Pearce, R.; Huang, X.; Setiawan, D.; Zhang, Y. EvoDesign: Designing Protein–Protein Binding Interactions Using Evolutionary Interface Profiles in Conjunction with an Optimized Physical Energy Function. *J Mol Biol* 2019, *431* (13), 2467–2476. https://doi.org/10.1016/j.jmb.2019.02.028.

(22) Adihou, H.; Gopalakrishnan, R.; Förster, T.; Guéret, S. M.; Gasper, R.; Geschwindner, S.; García, C. C.; Karatas, H.; Pobbati, A. V.; Vazquez-Chantada, M.; Davey, P.; Wassvik, C. M.; Pang, J. K. S.; Soh, B. S.; Hong, W.; Chiarparin, E.; Schade, D.; Plowright, A. T.; Valeur, E.; Lemurell, M.; Grossmann, T. N.; Waldmann, H. A Protein Tertiary Structure Mimetic Modulator of the Hippo Signalling Pathway. Nat Commun 2020, 11 (1), 5425. https://doi.org/10.1038/s41467-020-19224-8.

(23) Chevalier, A.; Silva, D.-A.; Rocklin, G. J.; Hicks, D. R.; Vergara, R.; Murapa, P.; Bernard, S. M.; Zhang, L.; Lam, K.-H.; Yao, G.; Bahl, C. D.; Miyashita, S.-I.; Goreshnik, I.; Fuller, J. T.; Koday, M. T.; Jenkins, C. M.; Colvin, T.; Carter, L.; Bohn, A.; Bryan, C. M.; Fernández-Velasco, D. A.; Stewart, L.; Dong, M.; Huang, X.; Jin, R.; Wilson, I. A.; Fuller, D. H.; Baker, D. Massively Parallel de Novo Protein Design for Targeted Therapeutics. Nature 2017, *550* (7674), nature23912. https://doi.org/10.1038/nature23912.

(24) Stranges, P. B.; Kuhlman, B. A Comparison of Successful and Failed Protein Interface Designs Highlights the Challenges of Designing Buried Hydrogen Bonds. Protein Sci 2013, *22* (1), 74–82. https://doi.org/10.1002/pro.2187.

(25) Fallas, J. A.; Ueda, G.; Sheffler, W.; Nguyen, V.; McNamara, D. E.; Sankaran, B.; Pereira, J. H.; Parmeggiani, F.; Brunette, T. J.; Cascio, D.; Yeates, T. R.; Zwart, P.; Baker, D. Computational Design of Self-Assembling Cyclic Protein Homo-Oligomers. *Nat Chem* 2017, *9* (4), 353–360. https://doi.org/10.1038/nchem.2673.

(26) Boyken, S. E.; Chen, Z.; Groves, B.; Langan, R. A.; Oberdorfer, G.; Ford, A.; Gilmore, J. M.; Xu, C.; DiMaio, F.; Pereira, J. H.; Sankaran, B.; Seelig, G.; Zwart, P. H.; Baker, D. De Novo Design of Protein Homo-Oligomers with Modular Hydrogen-Bond Network–Mediated Specificity. Science 2016, *352* (6286), 680–687. https://doi.org/10.1126/science.aad8865.

(27) Guharoy, M.; Chakrabarti, P. Secondary Structure Based Analysis and Classification of Biological Interfaces: Identification of Binding Motifs in Protein–Protein Interactions. *Bioinformatics* 2007, *23* (15), 1909–1918. https://doi.org/10.1093/bioinformatics/btm274.

(28) Brunette, T. J.; Bick, M. J.; Hansen, J. M.; Chow, C. M.; Kollman, J. M.; Baker, D. Modular Repeat Protein Sculpting Using Rigid Helical Junctions. *P Natl Acad Sci Usa* 2020, 201908768. https://doi.org/10.1073/pnas.1908768*117*.

(29) Jacobs, T.; Williams, B.; Williams, T.; Xu, X.; Eletsky, A.; Federizon, J.; Szyperski, T.; Kuhlman, B. Design of Structurally Distinct Proteins Using Strategies Inspired by Evolution. Science 2016, 352 (6286), 687–690. https://doi.org/10.1126/science.aad8036.

(30) Zhou, J.; Panaitiu, A. E.; Grigoryan, G. A General-Purpose Protein Design Framework Based on Mining Sequence–Structure Relationships in Known Protein Structures. Proc National Acad Sci 2020, 117 (2), 1059–1068. https://doi.org/10.1073/pnas.1908723117.

(31) Laniado, J.; Yeates, T. O. A Complete Rule Set for Designing Symmetry Combination Materials from Protein Molecules. Proc National Acad Sci 2020, 202015183. https://doi.org/10.1073/pnas.2015183117.

(32) Lai, Y.-T.; Reading, E.; Hura, G. L.; Tsai, K.-L.; Laganowsky, A.; Asturias, F. J.; Tainer, J. A.; Robinson, C. V.; Yeates, T. O. Structure of a Designed Protein Cage That Self-Assembles into a Highly Porous Cube. *Nature Chemistry* 2014, *6* (12), nchem.2107.

https://doi.org/10.1038/nchem.2107.

(33) Cock, P. J. A.; Antao, T.; Chang, J. T.; Chapman, B. A.; Cox, C. J.; Dalke, A.; Friedberg, I.; Hamelryck, T.; Kauff, F.; Wilczynski, B.; Hoon, M. J. L. de. Biopython: Freely Available Python Tools for Computational Molecular Biology and Bioinformatics. Bioinformatics 2009, *25* (11), 1422–1423. https://doi.org/10.1093/bioinformatics/btp163.

(34) Harris, C. R.; Millman, K. J.; Walt, S. J. van der; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N. J.; Kern, R.; Picus, M.; Hoyer, S.; Kerkwijk, M. H. van; Brett, M.; Haldane, A.; Río, J. F. del; Wiebe, M.; Peterson, P.; Gérard-Marchant, P.; Sheppard, K.; Reddy, T.; Weckesser, W.; Abbasi, H.; Gohlke, C.; Oliphant, T. E. Array Programming with NumPy. Nature 2020, *585* (7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2.

(35) Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research* 2011, *12*, 2825--2830.

(36) Mitternacht, S. FreeSASA: An Open Source C Library for Solvent Accessible Surface Area Calculations. *F1000research* 2016, *5*, 189. https://doi.org/10.12688/f1000research.7931.1.

(37) Jiang, S.; Tovchigrechko, A.; Vakser, I. A. The Role of Geometric Complementarity in Secondary Structure Packing: A Systematic Docking Study. Protein Sci 2003, *12* (8), 1646–1651. https://doi.org/10.1110/ps.0304503.

(38) Padhorny, D.; Kazennov, A.; Zerbe, B. S.; Porter, K. A.; Xia, B.; Mottarella, S. E.; Kholodov, Y.; Ritchie, D. W.; Vajda, S.; Kozakov, D. Protein–Protein Docking by Fast Generalized Fourier Transforms on 5D Rotational Manifolds. Proc National Acad Sci 2016, *113* (30), E4286–E4293. https://doi.org/10.1073/pnas.1603929113.

(39) Marze, N. A.; Burman, S. S. R.; Sheffler, W.; Gray, J. J. Efficient Flexible Backbone Protein–Protein Docking for Challenging Targets. Bioinformatics 2018, *34* (20), 3461–3469. https://doi.org/10.1093/bioinformatics/bty355.

(40) Mintseris, J.; Pierce, B.; Wiehe, K.; Anderson, R.; Chen, R.; Weng, Z. Integrating Statistical Pair Potentials into Protein Complex Prediction. *Proteins Struct Funct Bioinform* 2007, *69* (3), 511–520. https://doi.org/10.1002/prot.21502.

(41) Dey, S.; Ritchie, D. W.; Levy, E. D. PDB-Wide Identification of Biological Assemblies from Conserved Quaternary Structure Geometry. *Nature Methods* 2018, *15* (1), 67. https://doi.org/10.1038/nmeth.4510.

(42) Krissinel, E.; Henrick, K. Inference of Macromolecular Assemblies from Crystalline State. J Mol Biol 2007, *372* (3), 774–797. https://doi.org/10.1016/j.jmb.2007.05.022.

(43) Bale, J. B.; Park, R. U.; Liu, Y.; Gonen, S.; Gonen, T.; Cascio, D.; King, N. P.; Yeates, T. O.; Baker, D. Structure of a Designed Tetrahedral Protein Assembly Variant Engineered to Have Improved Soluble Expression. Protein Sci 2015, 24 (10), 1695–1701. https://doi.org/10.1002/pro.2748.

# EPILOGUE

In this body of work, we explore different experimental, theoretical and computational avenues with the central goal of advancing the field of symmetric protein design. First, we introduce a novel approach for designing nanoscale protein cages. Then, for the first time, we lay out a complete rule set for constructing all the kinds of protein-based materials that can be created by combining two symmetric oligomers. Finally, we implement a user-friendly computational design tool that integrates the entire rule set to enable the construction of a universe of novel protein nanomaterials. Despite our contributions and other recent advances in molecular engineering, designing self-assembling protein architectures remains challenging.

In Chapter 2, we sought to address the limitations of the oligomer fusion and interface design methodologies with the coiled coil approach. Although differing degrees of success ultimately led to crucial design lessons, problems associated with linker flexibility have yet to be solved. Additionally, a non-negligible fraction of the design candidates exhibited low solubility, most likely as a result of non-specific aggregation or misfolding caused by the leucine rich coiled coils. Alternate linkers with increased rigidity, solubility and specificity could improve the reliability of this approach. Other known heterotypic associations such as heterotrimeric coiled coils and heteromeric helical bundles might constitute good candidates. Additionally, combining fusion approaches with punctual interface design could mitigate issues associated with linker flexibility while alleviating challenges inherent to *de novo* interface design. Alternatively, with the complete rule set now in hand, negative design could improve prospects for strategies that rely on non-rigid linkers. For instance, when combining a C2 dimer and a C3 trimer to construct a tetrahedral cage, alternate D3, O and I assemblies could be modeled in advance with carefully-selected mutations that would disfavor these outcomes. Specifically, in the crystal structure of the collapsed ccT23-1 design, adjacent dimers in the D3 assembly were found to be in close proximity. These could be redesigned to shift the equilibrium to favor the intended tetrahedral assembly state.

Many of the possible symmetry combinations can give rise to multiple symmetric outcomes, many of which have large rings sizes. However, our findings illustrate that design targets that have large rings and that are the result of a geometrically promiscuous symmetry combination, can form alternate assemblies when the linkage between component oligomers is somewhat flexible. Thus, improving the reliability of the coiled coil design approach could open the door to an array of novel materials that are currently considered as difficult design targets.

In Chapter 3, we note several caveats to the completeness of our symmetry combination rule set. The oligomeric building blocks are required to obey one of the ordinary point group symmetry types. While such cases dominate among natural protein oligomers, helical assemblies are apparent in nature and can be used in combination with other symmetric components to construct various types of materials. In addition, while we direct our attention to two-component SCMs, symmetric materials can also be built from a single type of oligomeric component. By designing a new mode of self-association between multiple copies of the same oligomer type, a new symmetry operation can be introduced, such that a new one-component material forms upon self-assembly. Further, a vast number of different design types are possible when combining more than two symmetric components. On a slightly separate note, our current system does not consider quasiperiodic packings like Penrose tilings. Extending our theoretical framework to account for all of these possibilities could enable the design of an even greater range of nanoscale protein materials.

In Chapter 4, we address the problem of designing interfaces in the context of symmetric assembly. Central to our fragment-based docking approach is the use of known protein-protein interaction motifs to generate poses that exhibit native-like interfacial backbone arrangements. Our current interface fragment database only contains pairs of five residue segments. While these fragments capture some of the essential elements of local secondary structure interaction, additional features that cover different aspects of quaternary structure could be considered. Future versions of our database might include several distinct fragment sets that capture a hierarchy of diverse interaction motifs.

While Nanohedra identifies favorable poses and provides valuable information regarding interfacial amino acid preferences, a final step of sequence design is required to produce a complete model of a new biomaterial. Hence, further integration with existing amino acid sequence design programs would enable a seamless design pipeline.

Nanohedra exploits the advantages of pre-calculation methods and hash tables, nevertheless the program can be sped up in numerous ways. For instance, transformations during the docking procedure are currently applied to the coordinates of both oligomeric components. With the appropriate operations, this step can be performed on a single component. Doing so also removes the need for rebuilding a balltree when searching for potential clashes for each of the sampled degrees of freedom. Substantial accelerations can also be achieved by executing parts of the program, such as all-to-all distance calculations and transformations, on a GPU. Further, to avoid sampling redundant configurations, a system that treats orientational degeneracies is set in place for many of the possible construction types.

Completing this system would prevent potential oversampling for the remaining unhandled cases and could reduce program execution times.

Other areas of improvement include accounting for fortuitous interfaces that can emerge upon symmetry expansion, optimizing the Nanohedra score and implementing a graphical user interface to enable an even wider group of users to explore the large space of possible symmetric materials. From a broader perspective, while Nanohedra is geared for the design of symmetric assemblies, our fragment-based docking strategy could help advance protein docking and interface design in wider contexts.

## APPENDIX

## Nanohedra Source Code

## LICENSE

## README

```
NANOHEDRA: A FRAGMENT-BASED PROTEIN DOCKING TOOL FOR CONSTRUCTING SYMMETRY COMBINATION MATERIALS


NANOHEDRA IS LICENSED UNDER THE MIT LICENSE (SEE: LICENSE)


NANOHEDRA SETUP

STEP 1 - COMPILE 'orient_oligomer.f'
cd orient
gfortran -o orient_oligomer orient_oligomer.f

STEP 2 - INSTALL 'FreeSASA 2.0.3'
go to: https://freesasa.github.io for a quick installation guide

STEP 3 - INSTALL the following Python modules that support Python 2.7
'sklearn' version 0.20.1 (https://scikit-learn.org/0.20)
'biopython' version 1.72 (https://biopython.org/wiki/Download)


RUNNING NANOHEDRA
to access the user manual page:
python nanohedra.py


NOTES
nanohedra currently only supports Python 2.7
nanohedra currently runs on Linux/Unix and Mac
```

## CODE AVAILABILITY

The entire Nanohedra source code is available at: https://github.com/nanohedra/nanohedra

## PYTHON SOURCE CODE

**main**

nanohedra.py

```
from classes.FragDock import dock
from classes.EulerLookup import EulerLookup
from classes.Fragment import *
from classes.SymEntry import *
from utils.SamplingUtils import get_degeneracy_matrices
from utils.CmdLineArgParseUtils import *
from utils.NanohedraManualUtils import *
from utils.ExpandAssemblyUtils import *
import sys
import os
import subprocess


# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"


def main():
    cmd_line_in_params = sys.argv

    if len(cmd_line_in_params) > 1 and cmd_line_in_params[1] == '-dock':

        # Parse Command Line Input
        sym_entry_number, pdb_dir1_path, pdb_dir2_path, rot_step_deg1, rot_step_deg2, master_outdir,
output_exp_assembly, output_uc, output_surrounding_uc, min_matched, init_match_type = get_docking_parameters(
cmd_line_in_params)

        # Master Log File
        master_log_filepath = master_outdir + "/nanohedra_master_logfile.txt"

        # Making Master Output Directory
        if not os.path.exists(master_outdir):
            os.makedirs(master_outdir)

        # Getting PDB1 File paths
        pdb1_filepaths = []
        for root1, dirs1, files1 in os.walk(pdb_dir1_path):
            for file1 in files1:
                if '.pdb' in file1:
                    pdb1_filepaths.append(pdb_dir1_path + "/" + file1)
        if len(pdb1_filepaths) == 0:
            master_log_file = open(master_log_filepath, "a+")
            master_log_file.write("\nCOULD NOT FIND PDB FILE(S) IN THE INPUT DIRECTORY SPECIFIED FOR OLIGOMER 1:\n")
            master_log_file.write("%s\n" % pdb_dir1_path)
            master_log_file.write("NANOHEDRA DOCKING RUN ENDED\n")
            master_log_file.close()
            sys.exit()

        # Getting PDB2 File paths
        pdb2_filepaths = []
        for root2, dirs2, files2 in os.walk(pdb_dir2_path):
            for file2 in files2:
                if '.pdb' in file2:
                    pdb2_filepaths.append(pdb_dir2_path + "/" + file2)
        if len(pdb2_filepaths) == 0:
            master_log_file = open(master_log_filepath, "a+")
            master_log_file.write("\nCOULD NOT FIND PDB FILE(S) IN THE INPUT DIRECTORY SPECIFIED FOR OLIGOMER 2:\n")
            master_log_file.write("%s\n" % pdb_dir2_path)
            master_log_file.write("NANOHEDRA DOCKING RUN ENDED\n")
            master_log_file.close()
            sys.exit()

        try:
            # Nanohedra.py Path
            main_script_dir = os.path.dirname(os.path.realpath(__file__))

            # Free SASA Executable Path
            free_sasa_exe_path = "/usr/local/bin/freesasa"
            sasa_assert_error_message = "Could not locate freesasa executable here: %s\n" \
                                        "FreeSASA might not be (correctly) installed.\n" \
                                        "Go to: https://freesasa.github.io for a quick " \
                                        "installation guide." % free_sasa_exe_path
            assert os.path.exists(free_sasa_exe_path), sasa_assert_error_message

            sasa_v_proc = subprocess.Popen(['%s' % free_sasa_exe_path, '--version'], stdout=subprocess.PIPE,
                                           stderr=subprocess.PIPE)
            (sasa_v_out, sasa_v_err) = sasa_v_proc.communicate()
            free_sasa_v = sasa_v_out.split("\n")[0]
            assert_free_sasa_v_message = free_sasa_v + " not supported.\nInstall supported version: " \
```

**nanohedra.py**

```
                                             "FreeSASA 2.0.3 at https://freesasa.github.io"
            assert free_sasa_v == "FreeSASA 2.0.3", assert_free_sasa_v_message

            # Orient Oligomer Fortran Executable Path
            orient_executable_path = main_script_dir + "/orient/orient_oligomer"
            orient_assert_error_message = "Could not locate orient_oligomer executable here: %s\n" \
                                          "Check README file for instructions on how to compile " \
                                          "orient_oligomer.f" % orient_executable_path
            assert os.path.exists(orient_executable_path), orient_assert_error_message
            orient_executable_dir = os.path.dirname(orient_executable_path)

            # Fragment Database Directory Paths
            monofrag_cluster_rep_dirpath = main_script_dir + "/fragment_database/
Top5MonoFragClustersRepresentativeCentered"
            ijk_intfrag_cluster_rep_dirpath = main_script_dir + "/fragment_database/
Top75percent_IJK_ClusterRepresentatives_1A"
            intfrag_cluster_info_dirpath = main_script_dir + "/fragment_database/
IJK_ClusteredInterfaceFragmentDBInfo_1A"

            # SymEntry Parameters
            sym_entry = SymEntry(sym_entry_number)

            oligomer_symmetry_1 = sym_entry.get_group1_sym()
            oligomer_symmetry_2 = sym_entry.get_group2_sym()
            design_symmetry = sym_entry.get_pt_grp_sym()

            rot_range_deg_pdb1 = sym_entry.get_rot_range_deg_1()
            rot_range_deg_pdb2 = sym_entry.get_rot_range_deg_2()

            set_mat1 = sym_entry.get_rot_set_mat_group1()
            set_mat2 = sym_entry.get_rot_set_mat_group2()

            is_internal_zshift1 = sym_entry.is_internal_tx1()
            is_internal_zshift2 = sym_entry.is_internal_tx2()

            is_internal_rot1 = sym_entry.is_internal_rot1()
            is_internal_rot2 = sym_entry.is_internal_rot2()

            design_dim = sym_entry.get_design_dim()

            ref_frame_tx_dof1 = sym_entry.get_ref_frame_tx_dof_group1()
            ref_frame_tx_dof2 = sym_entry.get_ref_frame_tx_dof_group2()

            result_design_sym = sym_entry.get_result_design_sym()
            uc_spec_string = sym_entry.get_uc_spec_string()

            # Default Fragment Guide Atom Overlap Z-Value Threshold For Initial Matches
            init_max_z_val = 1.0

            # Default Fragment Guide Atom Overlap Z-Value Threshold For All Subsequent Matches
            subseq_max_z_val = 2.0

            master_log_file = open(master_log_filepath, "a+")

            # Default Rotation Step
            if is_internal_rot1 and rot_step_deg1 is None:
                rot_step_deg1 = 3  # If rotation step not provided but required, set rotation step to default
            if is_internal_rot2 and rot_step_deg2 is None:
                rot_step_deg2 = 3  # If rotation step not provided but required, set rotation step to default

            if not is_internal_rot1 and rot_step_deg1 is not None:
                rot_step_deg1 = 1
                master_log_file.write(
                    "Warning: Rotation Step 1 Specified Was Ignored. Oligomer 1 Does Not Have Internal Rotational DOF
\n\n")
            if not is_internal_rot2 and rot_step_deg2 is not None:
                rot_step_deg2 = 1
                master_log_file.write(
                    "Warning: Rotation Step 2 Specified Was Ignored. Oligomer 2 Does Not Have Internal Rotational DOF
\n\n")

            if not is_internal_rot1 and rot_step_deg1 is None:
                rot_step_deg1 = 1
            if not is_internal_rot2 and rot_step_deg2 is None:
                rot_step_deg2 = 1

            # For SCMs where the two oligomeric components have the same point group symmetry
            # make sure that there is at least 2 PDB files in the input PDB directory
            if (oligomer_symmetry_1 == oligomer_symmetry_2) and len(pdb1_filepaths) < 2:
                master_log_file.write("\nAT LEAST 2 OLIGOMERS ARE REQUIRED TO BE IN THE SPECIFIED INPUT DIRECTORY,\n"
)
```

nanohedra.py

```
                master_log_file.write("WHEN THE 2 COMPONENTS OF A SCM OBEY THE SAME POINT GROUP SYMMETRY ")
                master_log_file.write("(IN THIS CASE %s)\n" % oligomer_symmetry_1)
                master_log_file.write("%s PDB FILE(S) FOUND IN: %s\n" % (str(len(pdb1_filepaths)), pdb_dir1_path))
                master_log_file.write("NANOHEDRA DOCKING RUN ENDED\n")
                master_log_file.close()
                sys.exit()

            master_log_file.write("NANOHEDRA PROJECT INFORMATION\n")
            master_log_file.write("Oligomer 1 Input Directory: %s\n" % pdb_dir1_path)
            master_log_file.write("Oligomer 2 Input Directory: %s\n" % pdb_dir2_path)
            master_log_file.write("Master Output Directory: %s\n\n" % master_outdir)

            master_log_file.write("SYMMETRY COMBINATION MATERIAL INFORMATION\n")
            master_log_file.write("Nanohedra Entry Number: %s\n" % str(sym_entry_number))
            master_log_file.write("Oligomer 1 Point Group Symmetry: %s\n" % oligomer_symmetry_1)
            master_log_file.write("Oligomer 2 Point Group Symmetry: %s\n" % oligomer_symmetry_2)
            master_log_file.write("SCM Point Group Symmetry: %s\n" % design_symmetry)
            master_log_file.write("Oligomer 1 Internal ROT DOF: %s\n" % str(sym_entry.get_internal_rot1()))
            master_log_file.write("Oligomer 2 Internal ROT DOF: %s\n" % str(sym_entry.get_internal_rot2()))
            master_log_file.write("Oligomer 1 Internal Tx DOF: %s\n" % str(sym_entry.get_internal_tx1()))
            master_log_file.write("Oligomer 2 Internal Tx DOF: %s\n" % str(sym_entry.get_internal_tx2()))
            master_log_file.write("Oligomer 1 Setting Matrix: %s\n" % str(set_mat1))
            master_log_file.write("Oligomer 2 Setting Matrix: %s\n" % str(set_mat2))
            master_log_file.write("Oligomer 1 Reference Frame Tx DOF: %s\n" % (str(ref_frame_tx_dof1) if sym_entry.
is_ref_frame_tx_dof1() else str(None)))
            master_log_file.write("Oligomer 2 Reference Frame Tx DOF: %s\n" % (str(ref_frame_tx_dof2) if sym_entry.
is_ref_frame_tx_dof2() else str(None)))
            master_log_file.write("Resulting SCM Symmetry: %s\n" % result_design_sym)
            master_log_file.write("SCM Dimension: %s\n" % str(design_dim))
            master_log_file.write("SCM Unit Cell Specification: %s\n\n" % uc_spec_string)

            master_log_file.write("ROTATIONAL SAMPLING INFORMATION\n")
            master_log_file.write("Oligomer 1 ROT Sampling Range: %s\n" % (str(rot_range_deg_pdb1) if
is_internal_rot1 else "N/A"))
            master_log_file.write("Oligomer 2 ROT Sampling Range: %s\n" % (str(rot_range_deg_pdb2) if
is_internal_rot2 else "N/A"))
            master_log_file.write("Oligomer 1 ROT Sampling Step: %s\n" % (str(rot_step_deg1) if is_internal_rot1 else
 "N/A"))
            master_log_file.write("Oligomer 2 ROT Sampling Step: %s\n\n" % (str(rot_step_deg2) if is_internal_rot2
else "N/A"))

            # Orient Input Oligomers to Canonical Orientation
            if oligomer_symmetry_1 == oligomer_symmetry_2:
                master_log_file.write("ORIENTING INPUT OLIGOMER PDB FILES\n")
                master_log_file.close()
                oriented_pdb1_outdir = master_outdir + "/" + oligomer_symmetry_1 + "_oriented"
                if not os.path.exists(oriented_pdb1_outdir):
                    os.makedirs(oriented_pdb1_outdir)
                pdb1_oriented_filepaths = []
                pdb2_oriented_filepaths = []
                for pdb1_path in pdb1_filepaths:
                    pdb1 = PDB()
                    pdb1.readfile(pdb1_path, remove_alt_location=True)
                    pdb1_filename = os.path.basename(pdb1_path)
                    try:
                        pdb1.orient(oligomer_symmetry_1, oriented_pdb1_outdir, orient_executable_dir)
                        pdb1_oriented_filepaths.append(oriented_pdb1_outdir + "/" + pdb1_filename)
                        master_log_file = open(master_log_filepath, 'a+')
                        master_log_file.write("oriented: %s\n" % pdb1_filename)
                        master_log_file.close()
                    except ValueError as val_err:
                        master_log_file = open(master_log_filepath, 'a+')
                        master_log_file.write(str(val_err))
                        master_log_file.close()
                    except RuntimeError as rt_err:
                        master_log_file = open(master_log_filepath, 'a+')
                        master_log_file.write(str(rt_err))
                        master_log_file.close()
                if len(pdb1_oriented_filepaths) == 0:
                    master_log_file = open(master_log_filepath, "a+")
                    master_log_file.write("\nCOULD NOT ORIENT OLIGOMER INPUT PDB FILES\n")
                    master_log_file.write(
                        "CHECK %s/orient_oligomer_log.txt FOR MORE INFORMATION\n" % oriented_pdb1_outdir)
                    master_log_file.write("NANOHEDRA DOCKING RUN ENDED\n")
                    master_log_file.close()
                    sys.exit()
                elif len(pdb1_oriented_filepaths) == 1:
                    master_log_file = open(master_log_filepath, "a+")
                    master_log_file.write("\nAT LEAST 2 OLIGOMERS ARE REQUIRED WHEN THE ")
                    master_log_file.write("2 OLIGOMERIC COMPONENTS OF A SCM OBEY THE SAME POINT GROUP SYMMETRY ")
                    master_log_file.write("(IN THIS CASE: %s)\n" % oligomer_symmetry_1)
```

**nanohedra.py**

```
                master_log_file.write("HOWEVER ONLY 1 INPUT OLIGOMER PDB FILE COULD BE ORIENTED\n")
                master_log_file.write(
                    "CHECK %s/orient_oligomer_log.txt FOR MORE INFORMATION\n" % oriented_pdb1_outdir)
                master_log_file.write("NANOHEDRA DOCKING RUN ENDED\n")
                master_log_file.close()
                sys.exit()
            else:
                master_log_file = open(master_log_filepath, "a+")
                master_log_file.write(
                    "Successfully Oriented %s out of the %s Oligomer Input PDB Files\n==> %s\n\n"
                    % (str(len(pdb1_oriented_filepaths)), str(len(pdb1_filepaths)), oriented_pdb1_outdir))
                master_log_file.close()
        else:
            master_log_file.write("ORIENTING OLIGOMER 1 INPUT PDB FILE(S)\n")
            master_log_file.close()
            oriented_pdb1_outdir = master_outdir + "/" + oligomer_symmetry_1 + "_oriented"
            if not os.path.exists(oriented_pdb1_outdir):
                os.makedirs(oriented_pdb1_outdir)
            pdb1_oriented_filepaths = []
            for pdb1_path in pdb1_filepaths:
                pdb1 = PDB()
                pdb1.readfile(pdb1_path, remove_alt_location=True)
                pdb1_filename = os.path.basename(pdb1_path)
                try:
                    pdb1.orient(oligomer_symmetry_1, oriented_pdb1_outdir, orient_executable_dir)
                    pdb1_oriented_filepaths.append(oriented_pdb1_outdir + "/" + pdb1_filename)
                    master_log_file = open(master_log_filepath, 'a+')
                    master_log_file.write("oriented: %s\n" % pdb1_filename)
                    master_log_file.close()
                except ValueError as val_err:
                    master_log_file = open(master_log_filepath, 'a+')
                    master_log_file.write(str(val_err))
                    master_log_file.close()
                except RuntimeError as rt_err:
                    master_log_file = open(master_log_filepath, 'a+')
                    master_log_file.write(str(rt_err))
                    master_log_file.close()
            if len(pdb1_oriented_filepaths) == 0:
                master_log_file = open(master_log_filepath, "a+")
                master_log_file.write("\nCOULD NOT ORIENT OLIGOMER 1 INPUT PDB FILE(S)\n")
                master_log_file.write(
                    "CHECK %s/orient_oligomer_log.txt FOR MORE INFORMATION\n" % oriented_pdb1_outdir)
                master_log_file.write("NANOHEDRA DOCKING RUN ENDED\n")
                master_log_file.close()
                sys.exit()
            else:
                master_log_file = open(master_log_filepath, "a+")
                master_log_file.write(
                    "Successfully Oriented %s out of the %s Oligomer 1 Input PDB File(s)\n==> %s\n"
                    % (str(len(pdb1_oriented_filepaths)), str(len(pdb1_filepaths)), oriented_pdb1_outdir))
                master_log_file.close()

            master_log_file = open(master_log_filepath, 'a+')
            master_log_file.write("\nORIENTING OLIGOMER 2 INPUT PDB FILE(S)\n")
            master_log_file.close()
            oriented_pdb2_outdir = master_outdir + "/" + oligomer_symmetry_2 + "_oriented"
            if not os.path.exists(oriented_pdb2_outdir):
                os.makedirs(oriented_pdb2_outdir)
            pdb2_oriented_filepaths = []
            for pdb2_path in pdb2_filepaths:
                pdb2 = PDB()
                pdb2.readfile(pdb2_path, remove_alt_location=True)
                pdb2_filename = os.path.basename(pdb2_path)
                try:
                    pdb2.orient(oligomer_symmetry_2, oriented_pdb2_outdir, orient_executable_dir)
                    pdb2_oriented_filepaths.append(oriented_pdb2_outdir + "/" + pdb2_filename)
                    master_log_file = open(master_log_filepath, 'a+')
                    master_log_file.write("oriented: %s\n" % pdb2_filename)
                    master_log_file.close()
                except ValueError as val_err:
                    master_log_file = open(master_log_filepath, 'a+')
                    master_log_file.write(str(val_err))
                    master_log_file.close()
                except RuntimeError as rt_err:
                    master_log_file = open(master_log_filepath, 'a+')
                    master_log_file.write(str(rt_err))
                    master_log_file.close()
            if len(pdb2_oriented_filepaths) == 0:
                master_log_file = open(master_log_filepath, "a+")
                master_log_file.write("\nCOULD NOT ORIENT OLIGOMER 2 INPUT PDB FILE(S)\n")
                master_log_file.write(
```

**nanohedra.py**

```
                "CHECK %s/orient_oligomer_log.txt FOR MORE INFORMATION\n" % oriented_pdb2_outdir)
            master_log_file.write("NANOHEDRA DOCKING RUN ENDED\n")
            master_log_file.close()
            sys.exit()
        else:
            master_log_file = open(master_log_filepath, "a+")
            master_log_file.write(
                "Successfully Oriented %s out of the %s Oligomer 2 Input PDB File(s)\n==> %s\n\n"
                % (str(len(pdb2_oriented_filepaths)), str(len(pdb2_filepaths)), oriented_pdb2_outdir))
            master_log_file.close()

    # Get Degeneracy Matrices
    master_log_file = open(master_log_filepath, "a+")
    master_log_file.write("SEARCHING FOR POSSIBLE DEGENERACIES\n")
    degeneracy_matrices_1, degeneracy_matrices_2 = get_degeneracy_matrices(oligomer_symmetry_1,
                                                                           oligomer_symmetry_2,
                                                                           design_dim,
                                                                           design_symmetry)
    if degeneracy_matrices_1 is None:
        master_log_file.write("No Degeneracies Found for Oligomer 1\n")
    elif len(degeneracy_matrices_1) == 1:
        master_log_file.write("1 Degeneracy Found for Oligomer 1\n")
        master_log_file.write("%s\n\n" % str(degeneracy_matrices_1[0]))
    else:
        master_log_file.write("%s Degeneracies Found for Oligomer 1\n" % str(len(degeneracy_matrices_1)))
        for degen_mat_1 in degeneracy_matrices_1:
            master_log_file.write("%s\n" % str(degen_mat_1))
        master_log_file.write("\n")

    if degeneracy_matrices_2 is None:
        master_log_file.write("No Degeneracies Found for Oligomer 2\n\n")
    elif len(degeneracy_matrices_2) == 1:
        master_log_file.write("1 Degeneracy Found for Oligomer 2\n")
        master_log_file.write("%s\n\n" % str(degeneracy_matrices_2[0]))
    else:
        master_log_file.write("%s Degeneracies Found for Oligomer 2\n" % str(len(degeneracy_matrices_2)))
        for degen_mat_2 in degeneracy_matrices_2:
            master_log_file.write("%s\n" % str(degen_mat_2))
        master_log_file.write("\n")

    master_log_file.write("LOADING COMPLETE INTERFACE FRAGMENT REPRESENTATIVES DATABASE\n")
    # Create fragment database for all ijk cluster representatives
    ijk_frag_db = FragmentDB(monofrag_cluster_rep_dirpath,
                             ijk_intfrag_cluster_rep_dirpath,
                             intfrag_cluster_info_dirpath)
    # Get complete IJK fragment representatives database dictionaries
    ijk_monofrag_cluster_rep_pdb_dict = ijk_frag_db.get_monofrag_cluster_rep_dict()
    ijk_intfrag_cluster_rep_dict = ijk_frag_db.get_intfrag_cluster_rep_dict()
    ijk_intfrag_cluster_info_dict = ijk_frag_db.get_intfrag_cluster_info_dict()

    if init_match_type == "1_2":
        master_log_file.write("RETRIEVING HELIX-STRAND INTERFACE FRAGMENT REPRESENTATIVES FOR INITIAL MATCH\n
\n")
        # Get Helix-Strand fragment representatives database dictionaries for initial interface fragment
matching
        init_monofrag_cluster_rep_pdb_dict_1 = {"1": ijk_monofrag_cluster_rep_pdb_dict["1"]}
        init_monofrag_cluster_rep_pdb_dict_2 = {"2": ijk_monofrag_cluster_rep_pdb_dict["2"]}
        init_intfrag_cluster_rep_dict = {"1": {"2": ijk_intfrag_cluster_rep_dict["1"]["2"]}}
        init_intfrag_cluster_info_dict = {"1": {"2": ijk_intfrag_cluster_info_dict["1"]["2"]}}
    elif init_match_type == "2_1":
        master_log_file.write("RETRIEVING STRAND-HELIX INTERFACE FRAGMENT REPRESENTATIVES FOR INITIAL MATCH\n
\n")
        # Get Strand-Helix fragment representatives database dictionaries for initial interface fragment
matching
        init_monofrag_cluster_rep_pdb_dict_1 = {"2": ijk_monofrag_cluster_rep_pdb_dict["2"]}
        init_monofrag_cluster_rep_pdb_dict_2 = {"1": ijk_monofrag_cluster_rep_pdb_dict["1"]}
        init_intfrag_cluster_rep_dict = {"2": {"1": ijk_intfrag_cluster_rep_dict["2"]["1"]}}
        init_intfrag_cluster_info_dict = {"2": {"1": ijk_intfrag_cluster_info_dict["2"]["1"]}}
    elif init_match_type == "2_2":
        master_log_file.write("RETRIEVING STRAND-STRAND INTERFACE FRAGMENT REPRESENTATIVES FOR INITIAL MATCH\
n\n")
        # Get Strand-Strand fragment representatives database dictionaries for initial interface fragment
matching
        init_monofrag_cluster_rep_pdb_dict_1 = {"2": ijk_monofrag_cluster_rep_pdb_dict["2"]}
        init_monofrag_cluster_rep_pdb_dict_2 = {"2": ijk_monofrag_cluster_rep_pdb_dict["2"]}
        init_intfrag_cluster_rep_dict = {"2": {"2": ijk_intfrag_cluster_rep_dict["2"]["2"]}}
        init_intfrag_cluster_info_dict = {"2": {"2": ijk_intfrag_cluster_info_dict["2"]["2"]}}
    else:
        master_log_file.write("RETRIEVING HELIX-HELIX INTERFACE FRAGMENT REPRESENTATIVES FOR INITIAL MATCH\n\
n")
        # Get Helix-Helix fragment representatives database dictionaries for initial interface fragment
```

**nanohedra.py**

```
matching
                init_monofrag_cluster_rep_pdb_dict_1 = {"1": ijk_monofrag_cluster_rep_pdb_dict["1"]}
                init_monofrag_cluster_rep_pdb_dict_2 = {"1": ijk_monofrag_cluster_rep_pdb_dict["1"]}
                init_intfrag_cluster_rep_dict = {"1": {"1": ijk_intfrag_cluster_rep_dict["1"]["1"]}}
                init_intfrag_cluster_info_dict = {"1": {"1": ijk_intfrag_cluster_info_dict["1"]["1"]}}
            master_log_file.close()

            # Initialize Euler Lookup Class
            eul_lookup = EulerLookup()

            # Get Expand Matrices
            if design_dim == 0:
                expand_matrices = get_ptgrp_sym_op(result_design_sym)
            elif design_dim == 2:
                expand_matrices = get_sg_sym_op(result_design_sym.upper())
            elif design_dim == 3:
                expand_matrices = get_sg_sym_op(result_design_sym)
            else:
                master_log_file = open(master_log_filepath, "a+")
                master_log_file.write(
                    "\n%s is an Invalid Design Dimension. The Only Valid Dimensions are: 0, 2, 3\n" % str(design_dim)
)
                master_log_file.close()
                sys.exit()

            master_log_file = open(master_log_filepath, "a+")
            master_log_file.write("\nSTARTING FRAGMENT-BASED SYMMETRY DOCKING PROTOCOL\n\n")
            master_log_file.close()

            if oligomer_symmetry_1 == oligomer_symmetry_2:
                n = len(pdb1_oriented_filepaths)
                for i in range(n - 1):
                    pdb1_oriented_path = pdb1_oriented_filepaths[i]
                    pdb1_oriented_filename = os.path.splitext(os.path.basename(pdb1_oriented_path))[0]

                    for j in range(i + 1, n):
                        pdb2_oriented_path = pdb1_oriented_filepaths[j]
                        pdb2_oriented_filename = os.path.splitext(os.path.basename(pdb2_oriented_path))[0]

                        master_log_file = open(master_log_filepath, "a+")
                        master_log_file.write("DOCKING %s | %s\n" % (pdb1_oriented_filename, pdb2_oriented_filename))
                        master_log_file.close()

                        dock(init_intfrag_cluster_rep_dict, ijk_intfrag_cluster_rep_dict,
                             init_monofrag_cluster_rep_pdb_dict_1, init_monofrag_cluster_rep_pdb_dict_2,
                             init_intfrag_cluster_info_dict, ijk_monofrag_cluster_rep_pdb_dict,
                             ijk_intfrag_cluster_info_dict, free_sasa_exe_path, master_outdir, pdb1_oriented_path,
                             pdb2_oriented_path, set_mat1, set_mat2, ref_frame_tx_dof1, ref_frame_tx_dof2,
                             is_internal_zshift1, is_internal_zshift2, result_design_sym, uc_spec_string,
                             design_dim, expand_matrices, eul_lookup, init_max_z_val, subseq_max_z_val,
                             degeneracy_matrices_1, degeneracy_matrices_2, rot_step_deg1,
                             rot_range_deg_pdb1, rot_step_deg2, rot_range_deg_pdb2, output_exp_assembly,
                             output_uc, output_surrounding_uc, min_matched)

                        master_log_file = open(master_log_filepath, "a+")
                        master_log_file.write(
                            "COMPLETE ==> %s\n\n" %
                            (master_outdir + "/" + pdb1_oriented_filename + "_" + pdb2_oriented_filename))
                        master_log_file.close()

            else:
                for pdb1_oriented_path in pdb1_oriented_filepaths:
                    pdb1_oriented_filename = os.path.splitext(os.path.basename(pdb1_oriented_path))[0]

                    for pdb2_oriented_path in pdb2_oriented_filepaths:
                        pdb2_oriented_filename = os.path.splitext(os.path.basename(pdb2_oriented_path))[0]

                        master_log_file = open(master_log_filepath, "a+")
                        master_log_file.write("DOCKING %s | %s\n" % (pdb1_oriented_filename, pdb2_oriented_filename))
                        master_log_file.close()

                        dock(init_intfrag_cluster_rep_dict, ijk_intfrag_cluster_rep_dict,
                             init_monofrag_cluster_rep_pdb_dict_1, init_monofrag_cluster_rep_pdb_dict_2,
                             init_intfrag_cluster_info_dict, ijk_monofrag_cluster_rep_pdb_dict,
                             ijk_intfrag_cluster_info_dict, free_sasa_exe_path, master_outdir, pdb1_oriented_path,
                             pdb2_oriented_path, set_mat1, set_mat2, ref_frame_tx_dof1, ref_frame_tx_dof2,
                             is_internal_zshift1, is_internal_zshift2, result_design_sym, uc_spec_string,
                             design_dim, expand_matrices, eul_lookup, init_max_z_val, subseq_max_z_val,
                             degeneracy_matrices_1, degeneracy_matrices_2, rot_step_deg1,
                             rot_range_deg_pdb1, rot_step_deg2, rot_range_deg_pdb2, output_exp_assembly,
                             output_uc, output_surrounding_uc, min_matched)
```

**nanohedra.py**

```
                        master_log_file = open(master_log_filepath, "a+")
                        master_log_file.write(
                            "COMPLETE ==> %s\n\n" %
                            (master_outdir + "/" + pdb1_oriented_filename + "_" + pdb2_oriented_filename))
                        master_log_file.close()

            master_log_file = open(master_log_filepath, "a+")
            master_log_file.write("\nCOMPLETED FRAGMENT-BASED SYMMETRY DOCKING PROTOCOL\n\n")
            master_log_file.write("DONE\n")
            master_log_file.close()
            return 0

        except KeyboardInterrupt:
            master_log_file = open(master_log_filepath, "a+")
            master_log_file.write("\nRun Ended By KeyboardInterrupt\n")
            master_log_file.close()
            sys.exit()

    elif len(cmd_line_in_params) > 1 and cmd_line_in_params[1] == '-query':
        query_mode(cmd_line_in_params)

    elif len(cmd_line_in_params) > 1 and cmd_line_in_params[1] == '-postprocess':
        postprocess_mode(cmd_line_in_params)

    else:
        print_usage()


if __name__ == "__main__":
    main()
```

**nanohedra.py**

**classes**

Atom.py

```
# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"


class Atom:
    def __init__(self, number, type, alt_location, residue_type, chain, residue_number, code_for_insertion, x, y, z,
occ, temp_fact, element_symbol, atom_charge):
        self.number = number
        self.type = type
        self.alt_location = alt_location
        self.residue_type = residue_type
        self.chain = chain
        self.residue_number = residue_number
        self.code_for_insertion = code_for_insertion
        self.x = x
        self.y = y
        self.z = z
        self.occ = occ
        self.temp_fact = temp_fact
        self.element_symbol = element_symbol
        self.atom_charge = atom_charge

    def __str__(self):
        # prints Atom in PDB format
        return "{:6s}{:5d} {:^4s}{:1s}{:3s} {:1s}{:4d}{:1s}   {:8.3f}{:8.3f}{:8.3f}{:6.2f}{:6.2f}          {:>2s}{:2s
}\n".format("ATOM", self.number, self.type, self.alt_location, self.residue_type, self.chain, self.residue_number,
self.code_for_insertion, self.x, self.y, self.z, self.occ, self.temp_fact, self.element_symbol, self.atom_charge)

    def is_backbone(self):
        # returns True if atom is part of the proteins backbone and False otherwise
        backbone_specific_atom_type = ["N", "CA", "C", "O"]
        if self.type in backbone_specific_atom_type:
            return True
        else:
            return False

    def is_CB(self, InclGlyCA=False):
        if InclGlyCA:
            return self.type == "CB" or (self.type == "CA" and self.residue_type == "GLY")
        else:
            return self.type == "CB" or (self.type == "H" and self.residue_type == "GLY")

    def is_CA(self):
        return self.type == "CA"

    def coords(self):
        return [self.x, self.y, self.z]

    def __eq__(self, other):
        return (self.number == other.number and self.chain == other.chain and self.type == other.type and self.
residue_type == other.residue_type)

    def get_number(self):
        return self.number

    def get_type(self):
        return self.type

    def get_alt_location(self):
        return self.alt_location

    def get_residue_type(self):
        return self.residue_type

    def get_chain(self):
        return self.chain

    def get_residue_number(self):
        return self.residue_number

    def get_code_for_insertion(self):
        return self.code_for_insertion

    def get_x(self):
        return self.x

    def get_y(self):
        return self.y
```

**classes/Atom.py**

```
    def get_z(self):
        return self.z

    def get_occ(self):
        return self.occ

    def get_temp_fact(self):
        return self.temp_fact

    def get_element_symbol(self):
        return self.element_symbol

    def get_atom_charge(self):
        return self.atom_charge
```

**classes/Atom.py**

PDB.py

```
from Atom import Atom
import subprocess
import os
from shutil import copyfile
from shutil import move


# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"


class PDB:
    def __init__(self):
        self.all_atoms = []  # python list of Atoms
        self.filepath = None  # PDB filepath if instance is read from PDB file
        self.chain_id_list = []  # list of unique chain IDs in PDB
        self.cb_coords = []
        self.bb_coords = []

    def set_all_atoms(self, atom_list):
        self.all_atoms = atom_list

    def set_chain_id_list(self, chain_id_list):
        self.chain_id_list = chain_id_list

    def set_filepath(self, filepath):
        self.filepath = filepath

    def get_all_atoms(self):
        return self.all_atoms

    def get_chain_id_list(self):
        return self.chain_id_list

    def get_filepath(self):
        return self.filepath

    def readfile(self, filepath, remove_alt_location=False):
        # reads PDB file and feeds PDB instance
        self.filepath = filepath

        f = open(filepath, "r")
        pdb = f.readlines()
        f.close()

        available_chain_ids = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', '
R',
                               'S', 'T',
                               'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', '
l',
                               'm', 'n',
                               'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '0', '1', '2', '3', '4']

        chain_ids = []
        multimodel = False
        start_of_new_model = False
        model_chain_index = -1
        model_chain_id = None
        curr_chain_id = None
        for line in pdb:
            line = line.rstrip()
            if line[0:5] == "MODEL":
                start_of_new_model = True
                multimodel = True
                model_chain_index += 1
                model_chain_id = available_chain_ids[model_chain_index]
            elif line[0:4] == "ATOM":
                number = int(line[6:11].strip())
                type = line[12:16].strip()
                alt_location = line[16:17].strip()
                residue_type = line[17:20].strip()
                if multimodel:
                    if line[21:22].strip() != curr_chain_id:
                        curr_chain_id = line[21:22].strip()
                        if not start_of_new_model:
                            model_chain_index += 1
                            model_chain_id = available_chain_ids[model_chain_index]
                    start_of_new_model = False
                    chain = model_chain_id
```

**classes/PDB.py**

```
                else:
                    chain = line[21:22].strip()
                residue_number = int(line[22:26].strip())
                code_for_insertion = line[26:27].strip()
                x = float(line[30:38].strip())
                y = float(line[38:46].strip())
                z = float(line[46:54].strip())
                occ = float(line[54:60].strip())
                temp_fact = float(line[60:66].strip())
                element_symbol = line[76:78].strip()
                atom_charge = line[78:80].strip()
                atom = Atom(number, type, alt_location, residue_type, chain, residue_number, code_for_insertion, x, y
, z, occ, temp_fact, element_symbol, atom_charge)
                if remove_alt_location:
                    if alt_location == "" or alt_location == "A":
                        if atom.chain not in chain_ids:
                            chain_ids.append(atom.chain)
                        self.all_atoms.append(atom)
                else:
                    if atom.chain not in chain_ids:
                        chain_ids.append(atom.chain)
                    self.all_atoms.append(atom)
        self.chain_id_list = chain_ids

    def read_atom_list(self, atom_list, store_cb_and_bb_coords=False):
        # reads a python list of Atoms and feeds PDB instance
        if store_cb_and_bb_coords:
            chain_ids = []
            for atom in atom_list:
                self.all_atoms.append(atom)
                if atom.is_backbone():
                    [x, y, z] = [atom.x, atom.y, atom.z]
                    self.bb_coords.append([x, y, z])
                if atom.is_CB(InclGlyCA=False):
                    [x, y, z] = [atom.x, atom.y, atom.z]
                    self.cb_coords.append([x, y, z])
                if atom.chain not in chain_ids:
                    chain_ids.append(atom.chain)
            self.chain_id_list = chain_ids
        else:
            chain_ids = []
            for atom in atom_list:
                self.all_atoms.append(atom)
                if atom.chain not in chain_ids:
                    chain_ids.append(atom.chain)
            self.chain_id_list = chain_ids

    def chain(self, chain_id):
        # returns a python list of Atoms containing the subset of Atoms in the PDB instance that belong to the
selected chain ID
        selected_atoms = []
        for atom in self.all_atoms:
            if atom.chain == chain_id:
                selected_atoms.append(atom)
        return selected_atoms

    def extract_all_coords(self):
        coords = []
        for atom in self.all_atoms:
            [x, y, z] = [atom.x, atom.y, atom.z]
            coords.append([x, y, z])
        return coords

    def extract_backbone_coords(self):
        coords = []
        for atom in self.all_atoms:
            if atom.is_backbone():
                [x, y, z] = [atom.x, atom.y, atom.z]
                coords.append([x, y, z])
        return coords

    def get_CA_atoms(self):
        ca_atoms = []
        for atom in self.all_atoms:
            if atom.is_CA():
                ca_atoms.append(atom)
        return ca_atoms

    def get_backbone_atoms(self):
        bb_atoms = []
        for atom in self.all_atoms:
```

**classes/PDB.py**

```
            if atom.is_backbone():
                bb_atoms.append(atom)
        return bb_atoms

    def get_CB_coords(self, ReturnWithCBIndices=False, InclGlyCA=False):
        coords = []
        cb_indices = []
        for i in range(len(self.all_atoms)):
            if self.all_atoms[i].is_CB(InclGlyCA=InclGlyCA):
                [x, y, z] = [self.all_atoms[i].x, self.all_atoms[i].y, self.all_atoms[i].z]
                coords.append([x, y, z])
                if ReturnWithCBIndices:
                    cb_indices.append(i)
        if ReturnWithCBIndices:
            return coords, cb_indices
        else:
            return coords

    def replace_coords(self, new_cords):
        for i in range(len(self.all_atoms)):
            self.all_atoms[i].x, self.all_atoms[i].y, self.all_atoms[i].z = new_cords[i][0], new_cords[i][1],
new_cords[i][2]

    def mat_vec_mul3(self, a, b):
        c = [0. for i in range(3)]
        for i in range(3):
            c[i] = 0.
            for j in range(3):
                c[i] += a[i][j] * b[j]
        return c

    def rotate_translate(self, rot, tx):
        for atom in self.all_atoms:
            coord = [atom.x, atom.y, atom.z]
            coord_rot = self.mat_vec_mul3(rot, coord)
            newX = coord_rot[0] + tx[0]
            newY = coord_rot[1] + tx[1]
            newZ = coord_rot[2] + tx[2]
            atom.x, atom.y, atom.z = newX, newY, newZ

    def write(self, out_path, cryst1=None):
        outfile = open(out_path, "w")
        if cryst1 is not None and isinstance(cryst1, str) and cryst1.startswith("CRYST1"):
            outfile.write(str(cryst1))
        for atom in self.all_atoms:
            outfile.write(str(atom))
        outfile.close()

    def orient(self, symm, output_dir, orient_executable_dir):
        valid_subunit_number = {"C2": 2, "C3": 3, "C4": 4, "C5": 5, "C6": 6,
                                "D2": 4, "D3": 6, "D4": 8, "D5": 10, "D6": 12,
                                "I": 60, "O": 24, "T": 12}

        number_of_subunits = len(self.chain_id_list)

        pdb_file_name = os.path.basename(self.filepath)

        if number_of_subunits != valid_subunit_number[symm]:
            orient_log = open('%s/%s' % (output_dir, 'orient_oligomer_log.txt'), 'a+')
            orient_log.write("%s\n Oligomer could not be oriented: It has %s subunits while %s are expected "
                             "for %s symmetry\n\n" % (pdb_file_name, str(number_of_subunits),
                                                       str(valid_subunit_number[symm]), symm))
            orient_log.close()

            raise ValueError('orient_oligomer could not orient %s '
                             'check %s/orient_oligomer_log.txt for more information\n' % (pdb_file_name, output_dir))

        if not os.path.exists(output_dir):
            os.makedirs(output_dir)

        if os.path.exists(orient_executable_dir + "/input.pdb"):
            os.remove(orient_executable_dir + "/input.pdb")
        if os.path.exists(orient_executable_dir + "/output.pdb"):
            os.remove(orient_executable_dir + "/output.pdb")

        copyfile('%s' % self.filepath, '%s/input.pdb' % orient_executable_dir)

        process = subprocess.Popen(['%s/orient_oligomer' % orient_executable_dir],
                                   stdin=subprocess.PIPE,
                                   stdout=subprocess.PIPE,
                                   stderr=subprocess.PIPE,
```

**classes/PDB.py**

```
                                    cwd=orient_executable_dir)
        in_symm_file = '%s/symm_files/%s_symm.txt' % (orient_executable_dir, symm)
        stdout, stderr = process.communicate(input=in_symm_file)
        stdout = pdb_file_name + stdout[28:]

        orient_log = open('%s/%s' % (output_dir, 'orient_oligomer_log.txt'), 'a+')
        orient_log.write(stdout)
        if stderr != '':
            orient_log.write(stderr + "\n")
        else:
            orient_log.write('\n')
        orient_log.close()

        if os.path.exists(orient_executable_dir + "/output.pdb") and os.stat(orient_executable_dir + "/output.pdb").
st_size != 0:
            move('%s/output.pdb' % orient_executable_dir, '%s/%s' % (output_dir, pdb_file_name))

        if os.path.exists(orient_executable_dir + "/input.pdb"):
            os.remove(orient_executable_dir + "/input.pdb")
        if os.path.exists(orient_executable_dir + "/output.pdb"):
            os.remove(orient_executable_dir + "/output.pdb")

        if not os.path.exists('%s/%s' % (output_dir, pdb_file_name)):
            raise RuntimeError('orient_oligomer could not orient %s '
                               'check %s/orient_oligomer_log.txt for more information\n' % (pdb_file_name, output_dir
))

    def get_surface_resdiue_info(self, free_sasa_exe_path, probe_radius=2.2, sasa_thresh=0):
        # only works for monomers or homo-oligomers
        assert_error_message = "Could not locate freesasa executable here: %s" % free_sasa_exe_path
        assert os.path.exists(free_sasa_exe_path), assert_error_message

        proc = subprocess.Popen([free_sasa_exe_path, '--format=seq', '--probe-radius', str(probe_radius), self.
filepath]
                                , stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        (out, err) = proc.communicate()
        out_lines = out.split("\n")
        sasa_out = []
        for line in out_lines:
            if line != "\n" and line != "" and not line.startswith("#"):
                chain_id = line[4:5]
                res_num = int(line[5:10])
                sasa = float(line[17:])
                if sasa > sasa_thresh:
                    sasa_out.append((chain_id, res_num))
        return sasa_out
```

SymEntry.py

```python
# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"

# SYMMETRY COMBINATION MATERIAL TABLE (T.O.Y and J.L, 2020)
sym_comb_dict = {
    1: [1, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 2, '<0,0,0>', 'C2', 1, ['r:<0,0,1,c>', 't:<0,0,d>'], 1, '<0,0,0>',
'D2', 'D2', 0, 'N/A', 4, 2],
    2: [2, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,0,0>', 'C3', 2, ['r:<0,0,1,c>'], 1, '<e,0.577350*e,0>', 'C6'
, 'p6', 2, '(2*e, 2*e), 120', 4, 6],
    3: [3, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 2, '<0,0,0>', 'C3', 2, ['r:<0,0,1,c>', 't:<0,0,d>'], 1, '<0,0,0>',
'D3', 'D3', 0, 'N/A', 4, 2],
    4: [4, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 6, '<e,0,0>', 'C3', 2, ['r:<0,0,1,c>', 't:<0,0,d>'], 1, '<0,0,0>',
'D3', 'p312', 2, '(2*e, 2*e), 120', 5, 6],
    5: [5, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<0,0,0>', 'C3', 2, ['r:<0,0,1,c>', 't:<0,0,d>'], 4, '<0,0,0>',
'T', 'T', 0, 'N/A', 4, 3],
    6: [6, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<0,e,0>', 'C3', 2, ['r:<0,0,1,c>', 't:<0,0,d>'], 4, '<0,0,0>',
'T', 'I213', 3, '(4*e, 4*e, 4*e), (90, 90, 90)', 5, 10],
    7: [7, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 3, '<0,0,0>', 'C3', 2, ['r:<0,0,1,c>', 't:<0,0,d>'], 4, '<0,0,0>',
'O', 'O', 0, 'N/A', 4, 4],
    8: [8, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 3, '<2*e,e,0>', 'C3', 2, ['r:<0,0,1,c>', 't:<0,0,d>'], 4, '<0,0,0>'
, 'O', 'P4132', 3, '(8*e, 8*e, 8*e), (90, 90, 90)', 5, 10],
    9: [9, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<0,0,0>', 'C3', 2, ['r:<0,0,1,c>', 't:<0,0,d>'], 7, '<0,0,0>',
'I', 'I', 0, 'N/A', 4, 5],
    10: [10, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,0,0>', 'C4', 3, ['r:<0,0,1,c>'], 1, '<0,0,0>', 'C4', 'p4',
 2, '(2*e, 2*e), 90', 4, 4],
    11: [11, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 2, '<0,0,0>', 'C4', 3, ['r:<0,0,1,c>', 't:<0,0,d>'], 1, '<0,0,0>'
, 'D4', 'D4', 0, 'N/A', 4, 2],
    12: [12, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 8, '<0,0,0>', 'C4', 3, ['r:<0,0,1,c>', 't:<0,0,d>'], 1, '<e,0,0>'
, 'D4', 'p4212', 2, '(2*e, 2*e), 90', 5, 4],
    13: [13, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 3, '<0,0,0>', 'C4', 3, ['r:<0,0,1,c>', 't:<0,0,d>'], 1, '<0,0,0>'
, 'O', 'O', 0, 'N/A', 4, 3],
    14: [14, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 3, '<2*e,e,0>', 'C4', 3, ['r:<0,0,1,c>', 't:<0,0,d>'], 1, '<0,0,0
>', 'O', 'I432', 3, '(4*e, 4*e, 4*e), (90, 90, 90)', 5, 8],
    15: [15, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 2, '<0,0,0>', 'C5', 4, ['r:<0,0,1,c>', 't:<0,0,d>'], 1, '<0,0,0>'
, 'D5', 'D5', 0, 'N/A', 4, 2],
    16: [16, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<0,0,0>', 'C5', 4, ['r:<0,0,1,c>', 't:<0,0,d>'], 9, '<0,0,0>'
, 'I', 'I', 0, 'N/A', 4, 3],
    17: [17, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,0,0>', 'C6', 5, ['r:<0,0,1,c>'], 1, '<0,0,0>', 'C6', 'p6',
 2, '(2*e, 2*e), 120', 4, 3],
    18: [18, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 2, '<0,0,0>', 'C6', 5, ['r:<0,0,1,c>', 't:<0,0,d>'], 1, '<0,0,0>'
, 'D6', 'D6', 0, 'N/A', 4, 2],
    19: [19, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 6, '<e,0,0>', 'C6', 5, ['r:<0,0,1,c>', 't:<0,0,d>'], 1, '<0,0,0>'
, 'D6', 'p622', 2, '(2*e, 2*e), 120', 5, 4],
    20: [20, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,f,0>', 'D2', 6, ['None'], 1, '<0,0,0>', 'D2', 'c222', 2,
'(4*e, 4*f), 90', 4, 4],
    21: [21, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 8, '<0,0,0>', 'D2', 6, ['None'], 1, '<e,0,0>', 'D4', 'p422', 2,
'(2*e, 2*e), 90', 3, 4],
    22: [22, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 2, '<0,e,f>', 'D2', 6, ['None'], 5, '<0,0,0>', 'D4', 'I4122', 3,
'(4*e, 4*e, 8*f), (90, 90, 90)', 4, 6],
    23: [23, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 10, '<0,0,0>', 'D2', 6, ['None'], 1, '<e,0,0>', 'D6', 'p622', 2,
'(2*e, 2*e), 120', 3, 3],
    24: [24, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 10, '<0,0,e>', 'D2', 6, ['None'], 1, '<f,0,0>', 'D6', 'P6222', 3,
 '(2*f, 2*f, 6*e), (90, 90, 120)', 4, 6],
    25: [25, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 3, '<0,0,0>', 'D2', 6, ['None'], 5, '<2*e,0,e>', 'O', 'I432', 3,
'(4*e, 4*e, 4*e), (90, 90, 90)', 3, 4],
    26: [26, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 3, '<-2*e,3*e,0>', 'D2', 6, ['None'], 5, '<0,2*e,e>', 'O', 'I4132
', 3, '(8*e, 8*e, 8*e), (90, 90, 90)', 3, 3],
    27: [27, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 6, '<e,0,0>', 'D3', 7, ['None'], 11, '<0,0,0>', 'D3', 'p312', 2,
'(2*e, 2*e), 120', 3, 3],
    28: [28, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 2, '<0,e,f>', 'D3', 7, ['None'], 1, '<0,0,0>', 'D3', 'R32', 3, '(
3.4641*e, 3.4641*e, 3*f), (90, 90, 120)', 4, 4],
    29: [29, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,0,0>', 'D3', 7, ['None'], 11, '<e,0.57735*e,0>', 'D6', '
p622', 2, '(2*e, 2*e), 120', 3, 2],
    30: [30, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 2, '<0,0,0>', 'D3', 7, ['None'], 11, '<e,0.57735*e,0>', 'D6', '
p622', 2, '(2*e, 2*e), 120', 3, 2],
    31: [31, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 2, '<0,0,0>', 'D3', 7, ['None'], 11, '<e,0.57735*e,f>', 'D6', '
P6322', 3, '(2*e, 2*e, 4*f), (90, 90, 120)', 4, 4],
    32: [32, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<0,0,0>', 'D3', 7, ['None'], 4, '<e,e,e>', 'O', 'F4132', 3,
'(8*e, 8*e, 8*e), (90, 90, 90)', 3, 3],
    33: [33, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<0,2*e,0>', 'D3', 7, ['None'], 4, '<e,e,e>', 'O', 'I4132', 3,
 '(8*e, 8*e, 8*e), (90, 90, 90)', 3, 2],
    34: [34, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 3, '<0,0,0>', 'D3', 7, ['None'], 4, '<e,e,e>', 'O', 'I432', 3, '(
4*e, 4*e, 4*e), (90, 90, 90)', 3, 4],
    35: [35, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 3, '<0,e,-2*e>', 'D3', 7, ['None'], 4, '<e,e,e>', 'O', 'I4132', 3
, '(8*e, 8*e, 8*e), (90, 90, 90)', 3, 2],
    36: [36, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 3, '<0,e,-2*e>', 'D3', 7, ['None'], 4, '<3*e,3*e,3*e>', 'O', '
P4132', 3, '(8*e, 8*e, 8*e), (90, 90, 90)', 3, 3],
    37: [37, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,0,0>', 'D4', 8, ['None'], 1, '<0,0,0>', 'D4', 'p422', 2,
'(2*e, 2*e), 90', 3, 2],
```

**classes/SymEntry.py**

```
    38: [38, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 2, '<0,e,0>', 'D4', 8, ['None'], 1, '<0,0,0>', 'D4', 'p422', 2,
'(2*e, 2*e), 90', 3, 2],
    39: [39, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 8, '<0,e,f>', 'D4', 8, ['None'], 1, '<0,0,0>', 'D4', 'I422', 3,
'(2*e, 2*e, 4*f), (90, 90, 90)', 4, 4],
    40: [40, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 3, '<0,0,0>', 'D4', 8, ['None'], 1, '<0,0,e>', 'O', 'P432', 3, '(
2*e, 2*e, 2*e), (90, 90, 90)', 3, 3],
    41: [41, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 3, '<2*e,e,0>', 'D4', 8, ['None'], 1, '<2*e,2*e,0>', 'O', 'I432',
 3, '(4*e, 4*e, 4*e), (90, 90, 90)', 3, 2],
    42: [42, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,0,0>', 'D6', 9, ['None'], 1, '<0,0,0>', 'D6', 'p622', 2,
'(2*e, 2*e), 120', 3, 2],
    43: [43, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 6, '<e,0,0>', 'D6', 9, ['None'], 1, '<0,0,0>', 'D6', 'p622', 2,
'(2*e, 2*e), 120', 3, 2],
    44: [44, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 6, '<e,0,f>', 'D6', 9, ['None'], 1, '<0,0,0>', 'D6', 'P622', 3,
'(2*e, 2*e, 2*f), (90, 90, 120)', 4, 4],
    45: [45, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,0,0>', 'T', 10, ['None'], 1, '<0,0,0>', 'T', 'P23', 3, '(2
*e, 2*e, 2*e), (90, 90, 90)', 3, 2],
    46: [46, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,e,0>', 'T', 10, ['None'], 1, '<0,0,0>', 'T', 'F23', 3, '(4
*e, 4*e, 4*e), (90, 90, 90)', 3, 3],
    47: [47, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 3, '<2*e,3*e,0>', 'T', 10, ['None'], 1, '<0,4*e,0>', 'O', 'F4132'
, 3, '(8*e, 8*e, 8*e), (90, 90, 90)', 3, 2],
    48: [48, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,0,0>', 'O', 11, ['None'], 1, '<0,0,0>', 'O', 'P432', 3, '(
2*e, 2*e, 2*e), (90, 90, 90)', 3, 2],
    49: [49, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,e,0>', 'O', 11, ['None'], 1, '<0,0,0>', 'O', 'F432', 3, '(
4*e, 4*e, 4*e), (90, 90, 90)', 3, 2],
    50: [50, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 3, '<e,0,0>', 'O', 11, ['None'], 1, '<0,0,0>', 'O', 'F432', 3, '(
2*e, 2*e, 2*e), (90, 90, 90)', 3, 2],
    51: [51, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 3, '<0,e,0>', 'O', 11, ['None'], 1, '<0,0,0>', 'O', 'P432', 3, '(
2*e, 2*e, 2*e), (90, 90, 90)', 3, 2],
    52: [52, 'C2', 1, ['r:<0,0,1,a>', 't:<0,0,b>'], 3, '<-e,e,e>', 'O', 11, ['None'], 1, '<0,0,0>', 'O', 'I432', 3,
'(4*e, 4*e, 4*e), (90, 90, 90)', 3, 2],
    53: [53, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<0,0,0>', 'C3', 2, ['r:<0,0,1,c>'], 1, '<e,0.57735*e,0>', 'C3
', 'p3', 2, '(2*e, 2*e), 120', 4, 3],
    54: [54, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 4, '<0,0,0>', 'C3', 2, ['r:<0,0,1,c>', 't:<0,0,d>'], 12, '<0,0,0
>', 'T', 'T', 0, 'N/A', 4, 2],
    55: [55, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 4, '<0,0,0>', 'C3', 2, ['r:<0,0,1,c>', 't:<0,0,d>'], 12, '<e,0,0
>', 'T', 'P213', 3, '(2*e, 2*e, 2*e), (90, 90, 90)', 5, 5],
    56: [56, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 4, '<0,0,0>', 'C4', 3, ['r:<0,0,1,c>', 't:<0,0,d>'], 1, '<0,0,0>'
, 'O', 'O', 0, 'N/A', 4, 2],
    57: [57, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 4, '<0,0,0>', 'C4', 3, ['r:<0,0,1,c>', 't:<0,0,d>'], 1, '<e,0,0>'
, 'O', 'F432', 3, '(2*e, 2*e, 2*e), (90, 90, 90)', 5, 6],
    58: [58, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 7, '<0,0,0>', 'C5', 4, ['r:<0,0,1,c>', 't:<0,0,d>'], 9, '<0,0,0>'
, 'I', 'I', 0, 'N/A', 4, 2],
    59: [59, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,0.57735*e,0>', 'C6', 5, ['r:<0,0,1,c>'], 1, '<0,0,0>', 'C6
', 'p6', 2, '(2*e, 2*e), 120', 4, 2],
    60: [60, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,0.57735*e,0>', 'D2', 6, ['None'], 1, '<e,0,0>', 'D6', '
p622', 2, '(2*e, 2*e), 120', 3, 2],
    61: [61, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 4, '<0,0,0>', 'D2', 6, ['None'], 1, '<e,0,0>', 'T', 'P23', 3, '(2
*e, 2*e, 2*e), (90, 90, 90)', 3, 3],
    62: [62, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 4, '<0,0,0>', 'D2', 6, ['None'], 3, '<e,0,e>', 'O', 'F432', 3, '(
4*e, 4*e, 4*e), (90, 90, 90)', 3, 3],
    63: [63, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 4, '<0,0,0>', 'D2', 6, ['None'], 3, '<2*e,e,0>', 'O', 'I4132', 3,
 '(8*e,8*e, 8*e), (90, 90, 90)', 3, 2],
    64: [64, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,0.57735*e,0>', 'D3', 7, ['None'], 11, '<0,0,0>', 'D3', '
p312', 2, '(2*e, 2*e), 120', 3, 2],
    65: [65, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,0.57735*e,0>', 'D3', 7, ['None'], 1, '<0,0,0>', 'D3', '
p321', 2, '(2*e, 2*e), 120', 3, 2],
    66: [66, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 12, '<4*e,0,0>', 'D3', 7, ['None'], 4, '<3*e,3*e,3*e>', 'O', '
P4132', 3, '(8*e, 8*e, 8*e), (90, 90, 90)', 3, 4],
    67: [67, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 4, '<0,0,0>', 'D4', 8, ['None'], 1, '<0,0,e>', 'O', 'P432', 3, '(
2*e, 2*e, 2*e), (90, 90, 90)', 3, 2],
    68: [68, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,0.57735*e,0>', 'D6', 9, ['None'], 1, '<0,0,0>', 'D6', '
p622', 2, '(2*e, 2*e), 120', 3, 2],
    69: [69, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 4, '<e,0,0>', 'T', 10, ['None'], 1, '<0,0,0>', 'T', 'F23', 3, '(2
*e, 2*e, 2*e), (90, 90, 90)', 3, 2],
    70: [70, 'C3', 2, ['r:<0,0,1,a>', 't:<0,0,b>'], 4, '<e,0,0>', 'O', 11, ['None'], 1, '<0,0,0>', 'O', 'F432', 3, '(
2*e, 2*e, 2*e), (90, 90, 90)', 3, 2],
    71: [71, 'C4', 3, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<0,0,0>', 'C4', 3, ['r:<0,0,1,c>'], 1, '<e,e,0>', 'C4', 'p4',
 2, '(2*e, 2*e), 90', 4, 2],
    72: [72, 'C4', 3, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<0,0,0>', 'C4', 3, ['r:<0,0,1,c>', 't:<0,0,d>'], 2, '<0,e,e>'
, 'O', 'P432', 3, '(2*e, 2*e, 2*e), (90, 90, 90)', 5, 4],
    73: [73, 'C4', 3, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<0,0,0>', 'D2', 6, ['None'], 1, '<e,0,0>', 'D4', 'p422', 2,
'(2*e, 2*e), 90', 3, 2],
    74: [74, 'C4', 3, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,0,0>', 'D2', 6, ['None'], 5, '<0,0,0>', 'D4', 'p4212', 2,
'(2*e, 2*e), 90', 3, 2],
    75: [75, 'C4', 3, ['r:<0,0,1,a>', 't:<0,0,b>'], 2, '<0,0,0>', 'D2', 6, ['None'], 3, '<2*e,e,0>', 'O', 'I432', 3,
'(4*e, 4*e, 4*e), (90, 90, 90)', 3, 2],
    76: [76, 'C4', 3, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<0,0,0>', 'D2', 6, ['None'], 3, '<e,0,e>', 'O', 'F432', 3, '(
4*e, 4*e, 4*e), (90, 90, 90)', 3, 3],
    77: [77, 'C4', 3, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<0,0,0>', 'D3', 7, ['None'], 4, '<e,e,e>', 'O', 'I432', 3, '(
4*e, 4*e, 4*e), (90, 90, 90)', 3, 2],
    78: [78, 'C4', 3, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,e,0>', 'D4', 8, ['None'], 1, '<0,0,0>', 'D4', 'p422', 2,
```

**classes/SymEntry.py**

```
 '(2*e, 2*e), 90', 3, 2],
    79: [79, 'C4', 3, ['r:<0,0,1,a>', 't:<0,0,b>'], 2, '<0,0,0>', 'D4', 8, ['None'], 1, '<e,e,0>', 'O', 'P432', 3, '(
 2*e, 2*e, 2*e), (90, 90, 90)', 3, 2],
    80: [80, 'C4', 3, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<0,0,0>', 'T', 10, ['None'], 1, '<e,e,e>', 'O', 'F432', 3, '(
 4*e, 4*e, 4*e), (90, 90, 90)', 3, 2],
    81: [81, 'C4', 3, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<e,e,0>', 'O', 11, ['None'], 1, '<0,0,0>', 'O', 'P432', 3, '(
 2*e, 2*e, 2*e), (90, 90, 90)', 3, 2],
    82: [82, 'C6', 5, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<0,0,0>', 'D2', 6, ['None'], 1, '<e,0,0>', 'D6', 'p622', 2,
 '(2*e, 2*e), 120', 3, 2],
    83: [83, 'C6', 5, ['r:<0,0,1,a>', 't:<0,0,b>'], 1, '<0,0,0>', 'D3', 7, ['None'], 11, '<e,0.57735*e,0>', 'D6', '
 p622', 2, '(2*e, 2*e), 120', 2, 2],
    84: [84, 'D2', 6, ['None'], 1, '<0,0,0>', 'D2', 6, ['None'], 1, '<e,f,0>', 'D2', 'p222', 2, '(2*e, 2*f), 90', 2,
 2],
    85: [85, 'D2', 6, ['None'], 1, '<0,0,0>', 'D2', 6, ['None'], 1, '<e,f,g>', 'D2', 'F222', 3, '(4*e, 4*f, 4*g), (90
 , 90, 90)', 3, 3],
    86: [86, 'D2', 6, ['None'], 1, '<e,0,0>', 'D2', 6, ['None'], 5, '<0,0,f>', 'D4', 'P4222', 3, '(2*e, 2*e, 4*f), (
 90, 90, 90)', 2, 2],
    87: [87, 'D2', 6, ['None'], 1, '<e,0,0>', 'D2', 6, ['None'], 13, '<0,0,-f>', 'D6', 'P6222', 3, '(2*e, 2*e, 6*f
 ), (90, 90, 120)', 2, 2],
    88: [88, 'D2', 6, ['None'], 3, '<0,e,2*e>', 'D2', 6, ['None'], 5, '<0,2*e,e>', 'O', 'P4232', 3, '(4*e, 4*e, 4*e
 ), (90, 90, 90)', 1, 2],
    89: [89, 'D2', 6, ['None'], 1, '<e,0,0>', 'D3', 7, ['None'], 11, '<e,0.57735*e,0>', 'D6', 'p622', 2, '(2*e, 2*e
 ), 120', 1, 1],
    90: [90, 'D2', 6, ['None'], 1, '<e,0,0>', 'D3', 7, ['None'], 11, '<e,0.57735*e,f>', 'D6', 'P622', 3, '(2*e, 2*e,
 2*f), (90, 90, 120)', 2, 2],
    91: [91, 'D2', 6, ['None'], 1, '<0,0,2*e>', 'D3', 7, ['None'], 4, '<e,e,e>', 'D6', 'P4232', 3, '(4*e, 4*e, 4*e
 ), (90, 90, 90)', 1, 2],
    92: [92, 'D2', 6, ['None'], 3, '<2*e,e,0>', 'D3', 7, ['None'], 4, '<e,e,e>', 'O', 'I4132', 3, '(8*e, 8*e, 8*e), (
 90, 90, 90)', 1, 1],
    93: [93, 'D2', 6, ['None'], 1, '<e,0,0>', 'D4', 8, ['None'], 1, '<0,0,0>', 'D4', 'p422', 2, '(2*e, 2*e), 90', 1,
 1],
    94: [94, 'D2', 6, ['None'], 1, '<e,0,f>', 'D4', 8, ['None'], 1, '<0,0,0>', 'D4', 'P422', 3, '(2*e, 2*e, 2*f), (90
 , 90,90)', 2, 2],
    95: [95, 'D2', 6, ['None'], 5, '<e,0,f>', 'D4', 8, ['None'], 1, '<0,0,0>', 'D4', 'I422', 3, '(2*e, 2*e, 4*f), (90
 , 90,90)', 2, 2],
    96: [96, 'D2', 6, ['None'], 3, '<0,e,2*e>', 'D4', 8, ['None'], 1, '<0,0,2*e>', 'O', 'I432', 3, '(4*e, 4*e, 4*e
 ), (90, 90, 90)', 1, 1],
    97: [97, 'D2', 6, ['None'], 1, '<e,0,0>', 'D6', 9, ['None'], 1, '<0,0,0>', 'D6', 'p622', 2, '(2*e, 2*e), 120', 1,
  1],
    98: [98, 'D2', 6, ['None'], 1, '<e,0,f>', 'D6', 9, ['None'], 1, '<0,0,0>', 'D6', 'P622', 3, '(2*e, 2*e, 2*f), (90
 , 90, 120)', 2, 2],
    99: [99, 'D2', 6, ['None'], 1, '<e,0,0>', 'T', 10, ['None'], 1, '<0,0,0>', 'T', 'P23', 3, '(2*e, 2*e, 2*e), (90,
 90, 90)', 1, 1],
    100: [100, 'D2', 6, ['None'], 1, '<e,e,0>', 'T', 10, ['None'], 1, '<0,0,0>', 'T', 'P23', 3, '(2*e, 2*e, 2*e), (90
 , 90, 90)', 1, 2],
    101: [101, 'D2', 6, ['None'], 3, '<e,0,e>', 'T', 10, ['None'], 1, '<e,e,e>', 'O', 'F432', 3, '(4*e, 4*e, 4*e), (
 90, 90, 90)', 1, 1],
    102: [102, 'D2', 6, ['None'], 3, '<2*e,e,0>', 'T', 10, ['None'], 1, '<0,0,0>', 'O', 'P4232', 3, '(4*e, 4*e, 4*e
 ), (90, 90, 90)', 1, 2],
    103: [103, 'D2', 6, ['None'], 3, '<e,0,e>', 'O', 11, ['None'], 1, '<0,0,0>', 'O', 'F432', 3, '(4*e, 4*e, 4*e), (
 90, 90, 90)', 1, 1],
    104: [104, 'D2', 6, ['None'], 3, '<2*e,e,0>', 'O', 11, ['None'], 1, '<0,0,0>', 'O', 'I432', 3, '(4*e, 4*e, 4*e
 ), (90, 90, 90)', 1, 2],
    105: [105, 'D3', 7, ['None'], 11, '<0,0,0>', 'D3', 7, ['None'], 11, '<e,0.57735*e,0>', 'D3', 'p312', 2, '(2*e, 2*
 e), 120', 1, 1],
    106: [106, 'D3', 7, ['None'], 11, '<0,0,0>', 'D3', 7, ['None'], 11, '<e,0.57735*e,f>', 'D3', 'P312', 3, '(2*e, 2*
 e, 2*f), (90, 90, 120)', 2, 2],
    107: [107, 'D3', 7, ['None'], 1, '<0,0,0>', 'D3', 7, ['None'], 11, '<e,0.57735*e,f>', 'D6', 'P6322', 3, '(2*e, 2*
 e, 4*f), (90, 90, 120)', 2, 2],
    108: [108, 'D3', 7, ['None'], 4, '<e,e,e>', 'D3', 7, ['None'], 12, '<e,3*e,e>', 'O', 'P4232', 3, '(4*e, 4*e, 4*e
 ), (90, 90, 90)', 1, 2],
    109: [109, 'D3', 7, ['None'], 4, '<3*e,3*e,3*e>', 'D3', 7, ['None'], 12, '<e,3*e,5*e>', 'O', 'P4132', 3, '(8*e, 8
 *e, 8*e), (90, 90, 90)', 1, 1],
    110: [110, 'D3', 7, ['None'], 4, '<e,e,e>', 'D4', 8, ['None'], 1, '<0,0,2*e>', 'O', 'I432', 3, '(4*e, 4*e, 4*e
 ), (90, 90, 90)', 1, 2],
    111: [111, 'D3', 7, ['None'], 11, '<e,0.57735*e,0>', 'D6', 9, ['None'], 1, '<0,0,0>', 'D6', 'p622', 2, '(2*e, 2*e
 ), 120', 1, 1],
    112: [112, 'D3', 7, ['None'], 11, '<e,0.57735*e,f>', 'D6', 9, ['None'], 1, '<0,0,0>', 'D6', 'P622', 3, '(2*e, 2*e
 , 2*f), (90, 90, 120)', 2, 2],
    113: [113, 'D3', 7, ['None'], 4, '<e,e,e>', 'T', 10, ['None'], 1, '<0,0,0>', 'O', 'F4132', 3, '(8*e, 8*e, 8*e), (
 90, 90, 90)', 1, 1],
    114: [114, 'D3', 7, ['None'], 4, '<e,e,e>', 'O', 11, ['None'], 1, '<0,0,0>', 'O', 'I432', 3, '(4*e, 4*e, 4*e), (
 90, 90, 90)', 1, 1],
    115: [115, 'D4', 8, ['None'], 1, '<0,0,0>', 'D4', 8, ['None'], 1, '<e,e,0>', 'D4', 'p422', 2, '(2*e, 2*e), 90', 1
 , 1],
    116: [116, 'D4', 8, ['None'], 1, '<0,0,0>', 'D4', 8, ['None'], 1, '<e,e,f>', 'D4', 'P422', 3, '(2*e, 2*e, 2*f), (
 90, 90,90)', 2, 2],
    117: [117, 'D4', 8, ['None'], 1, '<0,0,e>', 'D4', 8, ['None'], 2, '<0,e,e>', 'O', 'P432', 3, '(2*e, 2*e, 2*e), (
 90, 90, 90)', 1, 1],
    118: [118, 'D4', 8, ['None'], 1, '<0,0,e>', 'O', 11, ['None'], 1, '<0,0,0>', 'O', 'P432', 3, '(2*e, 2*e, 2*e), (
 90, 90, 90)', 1, 1],
```

**classes/SymEntry.py**

```
    119: [119, 'D4', 8, ['None'], 1, '<e,e,0>', 'O', 11, ['None'], 1, '<0,0,0>', 'O', 'P432', 3, '(2*e, 2*e, 2*e), (
90, 90, 90)', 1, 1],
    120: [120, 'T', 10, ['None'], 1, '<0,0,0>', 'T', 10, ['None'], 1, '<e,e,e>', 'T', 'F23', 3, '(4*e, 4*e, 4*e), (90
, 90, 90)', 1, 1],
    121: [121, 'T', 10, ['None'], 1, '<0,0,0>', 'T', 10, ['None'], 1, '<e,0,0>', 'T', 'F23', 3, '(2*e, 2*e, 2*e), (90
, 90, 90)', 1, 1],
    122: [122, 'T', 10, ['None'], 1, '<e,e,e>', 'O', 11, ['None'], 1, '<0,0,0>', 'O', 'F432', 3, '(4*e, 4*e, 4*e), (
90, 90, 90)', 1, 1],
    123: [123, 'O', 11, ['None'], 1, '<0,0,0>', 'O', 11, ['None'], 1, '<e,e,e>', 'O', 'P432', 3, '(2*e, 2*e, 2*e), (
90, 90, 90)', 1, 1],
    124: [124, 'O', 11, ['None'], 1, '<0,0,0>', 'O', 11, ['None'], 1, '<e,0,0>', 'O', 'F432', 3, '(2*e, 2*e, 2*e), (
90, 90, 90)', 1, 1]}


# ROTATION RANGE DEG
C2 = 180
C3 = 120
C4 = 90
C5 = 72
C6 = 60
RotRangeDict = {"C2": C2, "C3": C3, "C4": C4, "C5": C5, "C6": C6}


# ROTATION SETTING MATRICES
RotMat1 = [[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]
RotMat2 = [[0.0, 0.0, 1.0], [0.0, 1.0, 0.0], [-1.0, 0.0, 0.0]]
RotMat3 = [[0.707107, 0.0, 0.707107], [0.0, 1.0, 0.0], [-0.707107, 0.0, 0.707107]]
RotMat4 = [[0.707107, 0.408248, 0.577350], [-0.707107, 0.408248, 0.577350], [0.0, -0.816497, 0.577350]]
RotMat5 = [[0.707107, 0.707107, 0.0], [-0.707107, 0.707107, 0.0], [0.0, 0.0, 1.0]]
RotMat6 = [[1.0, 0.0, 0.0], [0.0, 0.0, 1.0], [0.0, -1.0, 0.0]]
RotMat7 = [[1.0, 0.0, 0.0], [0.0, 0.934172, 0.356822], [0.0, -0.356822, 0.934172]]
RotMat8 = [[0.0, 0.707107, 0.707107], [0.0, -0.707107, 0.707107], [1.0, 0.0, 0.0]]
RotMat9 = [[0.850651, 0.0, 0.525732], [0.0, 1.0, 0.0], [-0.525732, 0.0, 0.850651]]
RotMat10 = [[0.0, 0.5, 0.866025], [0.0, -0.866025, 0.5], [1.0, 0.0, 0.0]]
RotMat11 = [[0.0, -1.0, 0.0], [1.0, 0.0, 0.0], [0.0, 0.0, 1.0]]
RotMat12 = [[0.707107, -0.408248, 0.577350], [0.707107, 0.408248, -0.577350], [0.0, 0.816497, 0.577350]]
RotMat13 = [[0.5, -0.866025, 0.0], [0.866025, 0.5, 0.0], [0.0, 0.0, 1.0]]

RotSetDict = {1: RotMat1,
              2: RotMat2,
              3: RotMat3,
              4: RotMat4,
              5: RotMat5,
              6: RotMat6,
              7: RotMat7,
              8: RotMat8,
              9: RotMat9,
              10: RotMat10,
              11: RotMat11,
              12: RotMat12,
              13: RotMat13}


class SymEntry:

    def __init__(self, entry):
        if type(entry) == int and entry in range(1, 125):
            # GETTING ENTRY INFORMATION FROM sym_comb_dict
            self.entry_number = entry
            sym_comb_info = sym_comb_dict[self.entry_number]

            # ASSIGNING CLASS VARIABLES
            self.group1 = sym_comb_info[1]
            self.group1_indx = sym_comb_info[2]
            self.int_dof_group1 = sym_comb_info[3]
            self.rot_set_group1 = sym_comb_info[4]
            self.ref_frame_tx_dof_group1 = sym_comb_info[5]
            self.group2 = sym_comb_info[6]
            self.group2_indx = sym_comb_info[7]
            self.int_dof_group2 = sym_comb_info[8]
            self.rot_set_group2 = sym_comb_info[9]
            self.ref_frame_tx_dof_group2 = sym_comb_info[10]
            self.pt_grp = sym_comb_info[11]
            self.result = sym_comb_info[12]
            self.dim = sym_comb_info[13]
            self.unit_cell = sym_comb_info[14]
            self.tot_dof = sym_comb_info[15]
            self.cycle_size = sym_comb_info[16]

        else:
            raise ValueError("\nINVALID SYMMETRY ENTRY. SUPPORTED VALUES ARE: 1 to 124\n")
```

**classes/SymEntry.py**

```
    def get_group1_sym(self):
        return self.group1

    def get_group2_sym(self):
        return self.group2

    def get_pt_grp_sym(self):
        return self.pt_grp

    def get_rot_range_deg_1(self):
        if self.group1 in RotRangeDict:
            return RotRangeDict[self.group1]
        else:
            return 0

    def get_rot_range_deg_2(self):
        if self.group2 in RotRangeDict:
            return RotRangeDict[self.group2]
        else:
            return 0

    def get_rot_set_mat_group1(self):
        return RotSetDict[self.rot_set_group1]

    def get_ref_frame_tx_dof_group1(self):
        return self.ref_frame_tx_dof_group1

    def get_rot_set_mat_group2(self):
        return RotSetDict[self.rot_set_group2]

    def get_ref_frame_tx_dof_group2(self):
        return self.ref_frame_tx_dof_group2

    def get_result_design_sym(self):
        return self.result

    def get_design_dim(self):
        return self.dim

    def get_uc_spec_string(self):
        return self.unit_cell

    def is_internal_tx1(self):
        if 't:<0,0,b>' in self.int_dof_group1:
            return True
        else:
            return False

    def is_internal_tx2(self):
        if 't:<0,0,d>' in self.int_dof_group2:
            return True
        else:
            return False

    def get_internal_tx1(self):
        if 't:<0,0,b>' in self.int_dof_group1:
            return 't:<0,0,b>'
        else:
            return None

    def get_internal_tx2(self):
        if 't:<0,0,d>' in self.int_dof_group2:
            return 't:<0,0,d>'
        else:
            return None

    def is_internal_rot1(self):
        if 'r:<0,0,1,a>' in self.int_dof_group1:
            return True
        else:
            return False

    def is_internal_rot2(self):
        if 'r:<0,0,1,c>' in self.int_dof_group2:
            return True
        else:
            return False

    def get_internal_rot1(self):
        if 'r:<0,0,1,a>' in self.int_dof_group1:
```

**classes/SymEntry.py**

```
            return 'r:<0,0,1,a>'
        else:
            return None

    def get_internal_rot2(self):
        if 'r:<0,0,1,c>' in self.int_dof_group2:
            return 'r:<0,0,1,c>'
        else:
            return None

    def is_ref_frame_tx_dof1(self):
        if self.ref_frame_tx_dof_group1 != '<0,0,0>':
            return True
        else:
            return False

    def is_ref_frame_tx_dof2(self):
        if self.ref_frame_tx_dof_group2 != '<0,0,0>':
            return True
        else:
            return False
```

**classes/SymEntry.py**

Fragment.py

```
from PDB import PDB
from Atom import Atom
from utils.BioPDBUtils import biopdb_aligned_chain
from utils.BioPDBUtils import biopdb_superimposer
import numpy as np
import sys
import os


# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"


def get_surface_fragments(pdb, free_sasa_exe_path):
    surface_frags = []
    surf_res_info = pdb.get_surface_resdiue_info(free_sasa_exe_path)

    for (chain, res_num) in surf_res_info:
        frag_atoms = []
        frag_res_nums = [res_num - 2, res_num - 1, res_num, res_num + 1, res_num + 2]
        ca_count = 0
        for atom in pdb.chain(chain):
            if atom.residue_number in frag_res_nums:
                frag_atoms.append(atom)
                if atom.is_CA():
                    ca_count += 1
        if ca_count == 5:
            surf_frag_pdb = PDB()
            surf_frag_pdb.read_atom_list(frag_atoms)
            surface_frags.append(surf_frag_pdb)

    return surface_frags


def get_surface_fragments_chain(pdb, chain_id, free_sasa_exe_path):
    surface_frags = []
    surf_res_info = pdb.get_surface_resdiue_info(free_sasa_exe_path)

    for (chain, res_num) in surf_res_info:
        if chain == chain_id:
            frag_atoms = []
            frag_res_nums = [res_num - 2, res_num - 1, res_num, res_num + 1, res_num + 2]
            ca_count = 0
            for atom in pdb.chain(chain):
                if atom.residue_number in frag_res_nums:
                    frag_atoms.append(atom)
                    if atom.is_CA():
                        ca_count += 1
            if ca_count == 5:
                surf_frag_pdb = PDB()
                surf_frag_pdb.read_atom_list(frag_atoms)
                surface_frags.append(surf_frag_pdb)

    return surface_frags


class GhostFragment:
    def __init__(self, pdb, i_frag_type, j_frag_type, k_frag_type, ghostfrag_central_res_tup,
aligned_surf_frag_central_res_tup, guide_atoms=None, guide_coords=None, pdb_coords=None):
        self.pdb = pdb
        self.i_frag_type = i_frag_type
        self.j_frag_type = j_frag_type
        self.k_frag_type = k_frag_type
        self.central_res_tup = ghostfrag_central_res_tup
        self.aligned_surf_frag_central_res_tup = aligned_surf_frag_central_res_tup

        if [guide_atoms, guide_coords, pdb_coords] == [None, None, None]:
            self.guide_atoms = []
            self.guide_coords = []
            self.pdb_coords = []
            for atom in self.pdb.all_atoms:
                self.pdb_coords.append([atom.x, atom.y, atom.z])
                if atom.chain == "9":
                    self.guide_atoms.append(atom)
                    self.guide_coords.append([atom.x, atom.y, atom.z])

        else:
            self.guide_atoms = guide_atoms
            self.guide_coords = guide_coords
```

**classes/Fragment.py**

```
            self.pdb_coords = pdb_coords

    def get_central_res_tup(self):
        return self.central_res_tup

    def get_aligned_surf_frag_central_res_tup(self):
        return self.aligned_surf_frag_central_res_tup

    def get_aligned_central_res_info(self):
        return self.central_res_tup + self.aligned_surf_frag_central_res_tup

    def get_i_frag_type(self):
        return self.i_frag_type

    def get_j_frag_type(self):
        return self.j_frag_type

    def get_k_frag_type(self):
        return self.k_frag_type

    def get_pdb(self):
        return self.pdb

    def get_pdb_coords(self):
        return self.pdb_coords

    def get_guide_atoms(self):
        return self.guide_atoms

    def get_guide_coords(self):
        return self.guide_coords

    def get_center_of_mass(self):
        return np.matmul(np.array([0.33333, 0.33333, 0.33333]), np.array(self.guide_coords))


class MonoFragment:
    def __init__(self, pdb, monofrag_cluster_rep_dict=None, type=None, guide_coords=None, central_res_num=None,
central_res_chain_id=None, pdb_coords=None, rmsd_thresh=0.75):
        self.pdb = None
        self.pdb_coords = None
        self.type = None
        self.guide_coords = None
        self.guide_atoms = None
        self.central_res_num = None
        self.central_res_chain_id = None

        if monofrag_cluster_rep_dict is None and type is not None and guide_coords is not None and central_res_num is
 not None and central_res_chain_id is not None and pdb_coords is not None:
            self.pdb = pdb
            self.pdb_coords = pdb_coords
            self.type = type
            self.guide_coords = guide_coords
            a1 = Atom(1, "CA", " ", "GLY", "9", 0, " ", guide_coords[0][0], guide_coords[0][1], guide_coords[0][2], 1
.00, 20.00, "C", "")
            a2 = Atom(2, "N", " ", "GLY", "9", 0, " ", guide_coords[1][0], guide_coords[1][1], guide_coords[1][2], 1.
00, 20.00, "N", "")
            a3 = Atom(3, "O", " ", "GLY", "9", 0, " ", guide_coords[2][0], guide_coords[2][1], guide_coords[2][2], 1.
00, 20.00, "O", "")
            self.guide_atoms = [a1, a2, a3]
            self.central_res_num = central_res_num
            self.central_res_chain_id = central_res_chain_id

        elif monofrag_cluster_rep_dict is not None and type is None and guide_coords is None and central_res_num is
None and central_res_chain_id is None and pdb_coords is None:
            self.pdb = pdb
            self.pdb_coords = self.pdb.extract_all_coords()
            frag_ca_atoms = self.pdb.get_CA_atoms()
            self.central_res_num = frag_ca_atoms[2].residue_number
            self.central_res_chain_id = self.pdb.chain_id_list[0]

            a1 = Atom(1, "CA", " ", "GLY", "9", 0, " ", 0.0, 0.0, 0.0, 1.00, 20.00, "C", "")
            a2 = Atom(2, "N", " ", "GLY", "9", 0, " ", 3.0, 0.0, 0.0, 1.00, 20.00, "N", "")
            a3 = Atom(3, "O", " ", "GLY", "9", 0, " ", 0.0, 3.0, 0.0, 1.00, 20.00, "O", "")

            min_rmsd = sys.maxint
            min_rmsd_cluster_rep_rot_tx = None
            min_rmsd_cluster_rep_type = None
            for cluster_type in monofrag_cluster_rep_dict:
                cluster_rep = monofrag_cluster_rep_dict[cluster_type]
                cluster_rep_ca_atoms = cluster_rep.get_CA_atoms()
```

**classes/Fragment.py**

```
                rmsd, rot, tx = biopdb_superimposer(frag_ca_atoms, cluster_rep_ca_atoms)

                if rmsd <= min_rmsd and rmsd <= rmsd_thresh:
                    min_rmsd = rmsd
                    min_rmsd_cluster_rep_rot_tx = rot, tx
                    min_rmsd_cluster_rep_type = cluster_type

            if min_rmsd_cluster_rep_rot_tx is not None:
                guide_atoms_pdb = PDB()
                guide_atoms_pdb.read_atom_list([a1, a2, a3])
                guide_atoms_pdb.rotate_translate(min_rmsd_cluster_rep_rot_tx[0], min_rmsd_cluster_rep_rot_tx[1])

                self.type = min_rmsd_cluster_rep_type
                self.guide_atoms = guide_atoms_pdb.all_atoms
                self.guide_coords = guide_atoms_pdb.extract_all_coords()

    def get_central_res_tup(self):
        return self.central_res_chain_id, self.central_res_num

    def get_guide_coords(self):
        return self.guide_coords

    def get_center_of_mass(self):
        if self.guide_coords is not None:
            return np.matmul(np.array([0.33333, 0.33333, 0.33333]), np.array(self.guide_coords))
        else:
            return None

    def get_type(self):
        return self.type

    def get_pdb(self):
        return self.pdb

    def get_pdb_coords(self):
        return self.pdb_coords

    def get_central_res_num(self):
        return self.central_res_num

    def get_central_res_chain_id(self):
        return self.central_res_chain_id

    def set_pdb(self, pdb):
        self.pdb = pdb
        self.pdb_coords = pdb.extract_all_coords()

    def set_guide_atoms(self, guide_coords):
        self.guide_coords = guide_coords
        a1 = Atom(1, "CA", " ", "GLY", "9", 0, " ", guide_coords[0][0], guide_coords[0][1], guide_coords[0][2], 1.00,
 20.00, "C", "")
        a2 = Atom(2, "N", " ", "GLY", "9", 0, " ", guide_coords[1][0], guide_coords[1][1], guide_coords[1][2], 1.00,
20.00, "N", "")
        a3 = Atom(3, "O", " ", "GLY", "9", 0, " ", guide_coords[2][0], guide_coords[2][1], guide_coords[2][2], 1.00,
20.00, "O", "")
        self.guide_atoms = [a1, a2, a3]

    def get_ghost_fragments(self, intfrag_cluster_rep_dict, kdtree_oligomer_backbone, clash_dist=2.2):
        if self.type in intfrag_cluster_rep_dict:
            ghost_fragments = []
            for j_type in intfrag_cluster_rep_dict[self.type]:
                for k_type in intfrag_cluster_rep_dict[self.type][j_type]:
                        intfrag = intfrag_cluster_rep_dict[self.type][j_type][k_type]
                        intfrag_pdb = intfrag[0]
                        intfrag_mapped_chain_id = intfrag[1]
                        intfrag_mapped_chain_central_res_num = intfrag[2]
                        intfrag_partner_chain_id = intfrag[3]
                        intfrag_partner_chain_central_res_num = intfrag[4]

                        aligned_ghost_frag_pdb = biopdb_aligned_chain(self.pdb, self.pdb.chain_id_list[0],
intfrag_pdb, intfrag_mapped_chain_id)

                        # Ghost Fragment Mapped Chain ID, Central Residue Number and Partner Chain ID, Partner
Central Residue Number
                        ghostfrag_central_res_tup = (intfrag_mapped_chain_id, intfrag_mapped_chain_central_res_num,
intfrag_partner_chain_id, intfrag_partner_chain_central_res_num)

                        # Only keep ghost fragments that don't clash with oligomer backbone
                        # Note: guide atoms, mapped chain atoms and non-backbone atoms not included
                        g_frag_bb_coords = []
```

**classes/Fragment.py**

```
                        for atom in aligned_ghost_frag_pdb.all_atoms:
                            if atom.chain != "9" and atom.chain != intfrag_mapped_chain_id and atom.is_backbone():
                                g_frag_bb_coords.append([atom.x, atom.y, atom.z])

                        cb_clash_count = kdtree_oligomer_backbone.two_point_correlation(g_frag_bb_coords, [clash_dist
])

                        if cb_clash_count[0] == 0:
                            ghost_fragments.append(GhostFragment(aligned_ghost_frag_pdb, self.type, j_type, k_type,
ghostfrag_central_res_tup, self.get_central_res_tup()))

            return ghost_fragments

        else:
            return None


class ClusterInfoFile:
    def __init__(self, infofile_path):
        self.infofile_path = infofile_path
        self.name = None
        self.size = None
        self.rmsd = None
        self.representative_filename = None
        self.central_residue_pair_freqs = []
        self.central_residue_pair_counts = []
        self.load_info()

    def load_info(self):
        infofile = open(self.infofile_path, "r")
        info_lines = infofile.readlines()
        infofile.close()
        is_res_freq_line = False
        for line in info_lines:

            if line.startswith("CLUSTER NAME:"):
                self.name = line.split()[2]
            if line.startswith("CLUSTER SIZE:"):
                self.size = int(line.split()[2])
            if line.startswith("CLUSTER RMSD:"):
                self.rmsd = float(line.split()[2])
            if line.startswith("CLUSTER REPRESENTATIVE NAME:"):
                self.representative_filename = line.split()[3]

            if line.startswith("CENTRAL RESIDUE PAIR COUNT:"):
                is_res_freq_line = False
            if is_res_freq_line:
                res_pair_type = (line.split()[0][0], line.split()[0][1])
                res_pair_freq = float(line.split()[1])
                self.central_residue_pair_freqs.append((res_pair_type, res_pair_freq))
            if line.startswith("CENTRAL RESIDUE PAIR FREQUENCY:"):
                is_res_freq_line = True

    def get_name(self):
        return self.name

    def get_size(self):
        return self.size

    def get_rmsd(self):
        return self.rmsd

    def get_representative_filename(self):
        return self.representative_filename

    def get_central_residue_pair_freqs(self):
        return self.central_residue_pair_freqs


class FragmentDB:
    def __init__(self, monofrag_cluster_rep_dirpath, intfrag_cluster_rep_dirpath, intfrag_cluster_info_dirpath):
        self.monofrag_cluster_rep_dirpath = monofrag_cluster_rep_dirpath
        self.intfrag_cluster_rep_dirpath = intfrag_cluster_rep_dirpath
        self.intfrag_cluster_info_dirpath = intfrag_cluster_info_dirpath

    def get_monofrag_cluster_rep_dict(self):
        cluster_rep_pdb_dict = {}
        for root, dirs, files in os.walk(self.monofrag_cluster_rep_dirpath):
            for filename in files:
                if filename.endswith(".pdb"):
                    pdb = PDB()
```

**classes/Fragment.py**

```
                    pdb.readfile(self.monofrag_cluster_rep_dirpath + "/" + filename, remove_alt_location=True)
                    cluster_rep_pdb_dict[os.path.splitext(filename)[0]] = pdb

        return cluster_rep_pdb_dict

    def get_intfrag_cluster_rep_dict(self):
        i_j_k_intfrag_cluster_rep_dict = {}
        for dirpath1, dirnames1, filenames1 in os.walk(self.intfrag_cluster_rep_dirpath):
            if not dirnames1:
                ijk_cluster_name = dirpath1.split("/")[-1]
                i_cluster_type = ijk_cluster_name.split("_")[0]
                j_cluster_type = ijk_cluster_name.split("_")[1]
                k_cluster_type = ijk_cluster_name.split("_")[2]

                if i_cluster_type not in i_j_k_intfrag_cluster_rep_dict:
                    i_j_k_intfrag_cluster_rep_dict[i_cluster_type] = {}

                if j_cluster_type not in i_j_k_intfrag_cluster_rep_dict[i_cluster_type]:
                    i_j_k_intfrag_cluster_rep_dict[i_cluster_type][j_cluster_type] = {}

                for dirpath2, dirnames2, filenames2 in os.walk(dirpath1):
                    for filename in filenames2:
                        if filename.endswith(".pdb"):
                            ijk_frag_cluster_rep_pdb = PDB()
                            ijk_frag_cluster_rep_pdb.readfile(dirpath1 + "/" + filename)
                            ijk_frag_cluster_rep_mapped_chain_id = filename[filename.find("mappedchain") + 12:
filename.find("mappedchain") + 13]
                            ijk_frag_cluster_rep_partner_chain_id = filename[filename.find("partnerchain") + 13:
filename.find("partnerchain") + 14]

                            # Get central residue number of mapped interface fragment chain
                            intfrag_mapped_chain_central_res_num = None
                            mapped_chain_res_count = 0
                            for atom in ijk_frag_cluster_rep_pdb.chain(ijk_frag_cluster_rep_mapped_chain_id):
                                if atom.is_CA():
                                    mapped_chain_res_count += 1
                                    if mapped_chain_res_count == 3:
                                        intfrag_mapped_chain_central_res_num = atom.residue_number

                            # Get central residue number of partner interface fragment chain
                            intfrag_partner_chain_central_res_num = None
                            partner_chain_res_count = 0
                            for atom in ijk_frag_cluster_rep_pdb.chain(ijk_frag_cluster_rep_partner_chain_id):
                                if atom.is_CA():
                                    partner_chain_res_count += 1
                                    if partner_chain_res_count == 3:
                                        intfrag_partner_chain_central_res_num = atom.residue_number

                            i_j_k_intfrag_cluster_rep_dict[i_cluster_type][j_cluster_type][k_cluster_type] = (
ijk_frag_cluster_rep_pdb, ijk_frag_cluster_rep_mapped_chain_id, intfrag_mapped_chain_central_res_num,
ijk_frag_cluster_rep_partner_chain_id, intfrag_partner_chain_central_res_num)

        return i_j_k_intfrag_cluster_rep_dict

    def get_intfrag_cluster_info_dict(self):
        intfrag_cluster_info_dict = {}
        for dirpath1, dirnames1, filenames1 in os.walk(self.intfrag_cluster_info_dirpath):
            if not dirnames1:
                ijk_cluster_name = dirpath1.split("/")[-1]
                i_cluster_type = ijk_cluster_name.split("_")[0]
                j_cluster_type = ijk_cluster_name.split("_")[1]
                k_cluster_type = ijk_cluster_name.split("_")[2]

                if i_cluster_type not in intfrag_cluster_info_dict:
                    intfrag_cluster_info_dict[i_cluster_type] = {}

                if j_cluster_type not in intfrag_cluster_info_dict[i_cluster_type]:
                    intfrag_cluster_info_dict[i_cluster_type][j_cluster_type] = {}

                for dirpath2, dirnames2, filenames2 in os.walk(dirpath1):
                    for filename in filenames2:
                        if filename.endswith(".txt"):
                            intfrag_cluster_info_dict[i_cluster_type][j_cluster_type][k_cluster_type] =
ClusterInfoFile(dirpath1 + "/" + filename)

        return intfrag_cluster_info_dict
```

**classes/Fragment.py**

EulerLookup.py

```python
import numpy as np
import os


# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"


class EulerLookup:
    def __init__(self, scale=3.0):

        nanohedra_dirpath = os.path.dirname(os.path.dirname(os.path.realpath(__file__)))
        binary_lookup_table_path = nanohedra_dirpath + "/euler_lookup/euler_lookup_40.npz"

        self.eul_lookup_40 = np.load(binary_lookup_table_path)['a']
        self.scale = scale

    @staticmethod
    def get_eulerint10_from_rot(rot):
        # convert rotation matrix to euler angles in the form of an integer triplet
        # (integer values are degrees divided by 10; these become indices for a lookup table)
        tolerance = 1.e-6
        eulint = np.zeros(3, dtype=int)
        rot[2, 2] = min(rot[2, 2], 1.)
        rot[2, 2] = max(rot[2, 2], -1.)

        # if |rot[2,2]|~1, let the 3rd angle (which becomes degernate with the 1st) be zero
        if rot[2, 2] > 1. - tolerance:
            e3 = 0.
            e1 = np.arctan2(rot[1, 0], rot[0, 0])
            e2 = 0.
        else:
            if rot[2, 2] < -(1. - tolerance):
                e3 = 0.
                e1 = np.arctan2(rot[1, 0], rot[0, 0])
                e2 = np.pi
            else:
                e2 = np.arccos(rot[2, 2])
                e1 = np.arctan2(rot[0, 2], -rot[1, 2])
                e3 = np.arctan2(rot[2, 0], rot[2, 1])

        eulint[0] = (np.rint(e1 * 180. / np.pi * 0.1 * 0.999999) + 36) % 36
        eulint[1] = np.rint(e2 * 180. / np.pi * 0.1 * 0.999999)
        eulint[2] = (np.rint(e3 * 180. / np.pi * 0.1 * 0.999999) + 36) % 36

        return eulint

    def get_eulint_from_guides(self, guide_ats):
        # take a set of guide atoms (3 xyz positions) and return integer indices
        # for the euler angles describing the orientations of the axes they form
        # Note that the positions are in a 3D array. Each guide_ats[i,:,:] is a
        # 3x3 array with the vectors stored *in columns*, i.e. one vector is in [i,:,j]
        # use known scale value to normalize, to save repeated sqrt calculations

        if guide_ats.ndim != 3 or guide_ats.shape[1] != 3 or guide_ats.shape[2] != 3:
            print ('ERROR: guide atom array with wrong dimensions')

        nfrags = guide_ats.shape[0]
        rot = np.zeros((3, 3))
        eulintarray = np.zeros((nfrags, 3), dtype=int)

        # form the 2 difference vectors, normalize, then cross product
        for i in range(nfrags):
            v1 = (guide_ats[i, :, 1] - guide_ats[i, :, 0]) * 1. / self.scale
            v2 = (guide_ats[i, :, 2] - guide_ats[i, :, 0]) * 1. / self.scale
            v3 = np.cross(v1, v2)
            rot = np.array([v1, v2, v3])

            # get the euler indices
            eulintarray[i, :] = self.get_eulerint10_from_rot(rot)

        return eulintarray

    def check_lookup_table(self, guide_coords_list1, guide_coords_list2):
        return_tup_list = []

        guide_list_1_np = np.array(guide_coords_list1)
        guide_list_1_np_T = np.array([atoms_coords_1.T for atoms_coords_1 in guide_list_1_np])
```

**classes/EulerLookup.py**

```
        guide_list_2_np = np.array(guide_coords_list2)
        guide_list_2_np_T = np.array([atoms_coords_2.T for atoms_coords_2 in guide_list_2_np])

        eulintarray1 = self.get_eulint_from_guides(guide_list_1_np_T)
        eulintarray2 = self.get_eulint_from_guides(guide_list_2_np_T)

        # check lookup table
        for i in range(len(eulintarray1)):
            for j in range(len(eulintarray2)):
                (e1, e2, e3) = eulintarray1[i, :].flatten()
                (f1, f2, f3) = eulintarray2[j, :].flatten()
                return_tup_list.append((i, j, self.eul_lookup_40[e1, e2, e3, f1, f2, f3]))

        return return_tup_list
```

**classes/EulerLookup.py**

OptimalTx.py

```
import numpy as np
import sys
from math import sqrt


# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"


class OptimalTx:
    def __init__(self, setting1, setting2, is_zshift1, is_zshift2, cluster_rmsd, guide_atom_coods1, guide_atom_coods2
, dof_ext):
        self.setting1 = np.array(setting1)
        self.setting2 = np.array(setting2)
        self.is_zshift1 = is_zshift1
        self.is_zshift2 = is_zshift2
        self.dof_ext = np.array(dof_ext)
        self.n_dof_external = len(self.dof_ext)
        self.cluster_rmsd = cluster_rmsd
        self.guide_atom_coods1 = guide_atom_coods1
        self.guide_atom_coods2 = guide_atom_coods2

        self.n_dof_internal = [self.is_zshift1, self.is_zshift2].count(True)
        self.optimal_tx = (np.array([]), sys.maxint)  # (shift, error_zvalue)
        self.guide_atom_coods1_set = []
        self.guide_atom_coods2_set = []

    def dof_convert9(self):
        # convert input degrees of freedom to 9-dim arrays
        ndof = len(self.dof_ext)
        dof = np.zeros((ndof, 9))
        for i in range(ndof):
            dof[i] = (np.array(3 * [self.dof_ext[i]])).flatten()
        return np.transpose(dof)

    def solve_optimal_shift(self):
        # This routine does the work to solve the optimal shift problem

        # form the guide atoms into a matrix (column vectors)
        guide_target_10 = np.transpose(np.array(self.guide_atom_coods1_set))
        guide_query_10 = np.transpose(np.array(self.guide_atom_coods2_set))

        # calculate the initial difference between query and target (9 dim vector)
        guide_delta = np.transpose([guide_target_10.flatten('F') - guide_query_10.flatten('F')])

        # isotropic case based on simple rmsd
        self.cluster_rmsd = max(self.cluster_rmsd, 0.01)
        diagval = 1. / (3. * self.cluster_rmsd ** 2)
        var_tot_inv = np.zeros([9, 9])
        for i in range(9):
            var_tot_inv[i, i] = diagval

        # add internal z-shift degrees of freedom to 9-dim arrays if they exist
        if self.is_zshift1:
            self.dof_ext = np.append(self.dof_ext, -self.setting1[:, 2:3].T, axis=0)
        if self.is_zshift2:
            self.dof_ext = np.append(self.dof_ext, self.setting2[:, 2:3].T, axis=0)

        # convert degrees of freedom to 9-dim array
        dof = self.dof_convert9()

        # solve the problem
        dofT = np.transpose(dof)
        dinvv = np.matmul(var_tot_inv, dof)
        vtdinvv = np.matmul(dofT, dinvv)
        vtdinvvinv = np.linalg.inv(vtdinvv)

        dinvdelta = np.matmul(var_tot_inv, guide_delta)
        vtdinvdelta = np.matmul(dofT, dinvdelta)

        shift = np.matmul(vtdinvvinv, vtdinvdelta)

        # get error value
        resid = np.matmul(dof, shift) - guide_delta
        residT = np.transpose(resid)

        error = sqrt(np.matmul(residT, resid) / float(3.0)) / self.cluster_rmsd  # sqrt(variance / 3) / cluster_rmsd
 # NEW ERROR
```

**classes/OptimalTx.py**

```
        self.optimal_tx = (shift[:, 0], error)  # (shift, error_zvalue)

    @staticmethod
    def mat_vec_mul3(a, b):
        c = [0. for i in range(3)]

        for i in range(3):
            c[i] = 0.
            for j in range(3):
                c[i] += a[i][j] * b[j]

        return c

    def set_guide_atoms(self, rot_mat, coords):
        rotated_coords = []

        for coord in coords:
            x, y, z = self.mat_vec_mul3(rot_mat, [coord[0], coord[1], coord[2]])
            rotated_coords.append([x, y, z])

        return rotated_coords

    def apply(self):
        # Apply Setting Matrix to Guide Atoms
        self.guide_atom_coods1_set = self.set_guide_atoms(self.setting1, self.guide_atom_coods1)
        self.guide_atom_coods2_set = self.set_guide_atoms(self.setting2, self.guide_atom_coods2)

        # solve for shifts and resulting error
        self.solve_optimal_shift()

    def get_optimal_tx_dof_int(self):
        tx_dof_int = []

        shift, error_zvalue = self.optimal_tx
        index = self.n_dof_external

        if self.is_zshift1:
            tx_dof_int.append(shift[index:index + 1][0])
            index += 1

        if self.is_zshift2:
            tx_dof_int.append(shift[index:index + 1][0])

        return tx_dof_int

    def get_optimal_tx_dof_ext(self):
        shift, error_zvalue = self.optimal_tx
        return shift[0:self.n_dof_external].tolist()

    def get_all_optimal_shifts(self):
        shift, error_zvalue = self.optimal_tx
        return shift.tolist()

    def get_n_dof_external(self):
        return self.n_dof_external

    def get_n_dof_internal(self):
        return self.n_dof_internal

    def get_zvalue(self):
        shift, error_zvalue = self.optimal_tx
        return error_zvalue
```

**classes/OptimalTx.py**

FragDock.py

```
import os
from classes.OptimalTx import *
from classes.Fragment import *
from classes.WeightedSeqFreq import FragMatchInfo
from classes.WeightedSeqFreq import SeqFreqInfo
from utils.GeneralUtils import *
from utils.SamplingUtils import *
from utils.PDBUtils import *
from utils.SymmUtils import get_uc_dimensions
from utils.ExpandAssemblyUtils import generate_cryst1_record
from utils.ExpandAssemblyUtils import expanded_design_is_clash
import math
import sklearn.neighbors
import numpy as np
import time


# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"


def write_frag_match_info_file(ghost_frag, surf_frag, z_value, cluster_id, match_count, res_freq_list,
                               cluster_rmsd, outdir_path, pose_id, match_number, is_initial_match=False):

    out_info_file_path = outdir_path + "/frag_match_info_file.txt"
    out_info_file = open(out_info_file_path, "a+")

    aligned_central_res_info = ghost_frag.get_aligned_central_res_info()
    surf_frag_oligomer2_central_res_tup = surf_frag.get_central_res_tup()

    if is_initial_match:
        out_info_file.write("DOCKED POSE ID: %s\n\n" % pose_id)
        out_info_file.write("***** INITIAL MATCH FROM REPRESENTATIVES OF INITIAL FRAGMENT CLUSTERS *****\n\n")

    out_info_file.write("MATCH %s\n" % str(match_number))
    out_info_file.write("z-val: %s\n" % str(z_value))
    out_info_file.write("CENTRAL RESIDUES\n")
    out_info_file.write("oligomer1 ch, resnum: %s, %s\n" %
                        (str(aligned_central_res_info[4]), str(aligned_central_res_info[5])))
    out_info_file.write("oligomer2 ch, resnum: %s, %s\n" %
                        (str(surf_frag_oligomer2_central_res_tup[0]), str(surf_frag_oligomer2_central_res_tup[1])))
    out_info_file.write("FRAGMENT CLUSTER\n")
    out_info_file.write("id: %s\n" % cluster_id)
    out_info_file.write("mean rmsd: %s\n" % str(cluster_rmsd))
    out_info_file.write("aligned rep: int_frag_%s_%s.pdb\n" % (cluster_id, str(match_count)))
    out_info_file.write("central res pair freqs:\n%s\n\n" % str(res_freq_list))

    if is_initial_match:
        out_info_file.write("***** ALL MATCH(ES) FROM REPRESENTATIVES OF ALL FRAGMENT CLUSTERS *****\n\n")

    out_info_file.close()


def write_docked_pose_info(outdir_path, res_lev_sum_score, high_qual_match_count,
                           unique_matched_interface_monofrag_count, unique_total_interface_monofrags_count,
                           percent_of_interface_covered, rot_mat1, representative_int_dof_tx_param_1, set_mat1,
                           representative_ext_dof_tx_params_1, rot_mat2, representative_int_dof_tx_param_2, set_mat2,
                           representative_ext_dof_tx_params_2, cryst1_record, pdb1_path, pdb2_path, pose_id):

    out_info_file_path = outdir_path + "/docked_pose_info_file.txt"
    out_info_file = open(out_info_file_path, "w")

    out_info_file.write("DOCKED POSE ID: %s\n\n" % pose_id)

    out_info_file.write("Nanohedra Score: %s\n\n" % str(res_lev_sum_score))

    out_info_file.write("Unique Mono Fragments Matched (z<=1): %s\n" % str(high_qual_match_count))
    out_info_file.write("Unique Mono Fragments Matched: %s\n" % str(unique_matched_interface_monofrag_count))
    out_info_file.write("Unique Mono Fragments at Interface: %s\n" % str(unique_total_interface_monofrags_count))
    out_info_file.write("Interface Matched (%s): %s\n\n" % ("%", str(percent_of_interface_covered * 100)))

    out_info_file.write("ROT/DEGEN MATRIX PDB1: %s\n" % str(rot_mat1))
    if representative_int_dof_tx_param_1 is not None:
        int_dof_tx_vec_1 = representative_int_dof_tx_param_1
    else:
        int_dof_tx_vec_1 = None
    out_info_file.write("INTERNAL Tx PDB1: " + str(int_dof_tx_vec_1) + "\n")
    out_info_file.write("SETTING MATRIX PDB1: " + str(set_mat1) + "\n")
    if representative_ext_dof_tx_params_1 == [0, 0, 0]:
```

classes/FragDock.py

```
        ref_frame_tx_vec_1 = None
    else:
        ref_frame_tx_vec_1 = representative_ext_dof_tx_params_1
    out_info_file.write("REFERENCE FRAME Tx PDB1: " + str(ref_frame_tx_vec_1) + "\n\n")

    out_info_file.write("ROT/DEGEN MATRIX PDB2: %s\n" % str(rot_mat2))
    if representative_int_dof_tx_param_2 is not None:
        int_dof_tx_vec_2 = representative_int_dof_tx_param_2
    else:
        int_dof_tx_vec_2 = None
    out_info_file.write("INTERNAL Tx PDB2: " + str(int_dof_tx_vec_2) + "\n")
    out_info_file.write("SETTING MATRIX PDB2: " + str(set_mat2) + "\n")
    if representative_ext_dof_tx_params_2 == [0, 0, 0]:
        ref_frame_tx_vec_2 = None
    else:
        ref_frame_tx_vec_2 = representative_ext_dof_tx_params_2
    out_info_file.write("REFERENCE FRAME Tx PDB2: " + str(ref_frame_tx_vec_2) + "\n\n")

    out_info_file.write("CRYST1 RECORD: %s\n\n" % str(cryst1_record))

    out_info_file.write('Canonical Orientation PDB1 Path: %s\n' % pdb1_path)
    out_info_file.write('Canonical Orientation PDB2 Path: %s\n\n' % pdb2_path)

    out_info_file.close()


def out(pdb1, pdb2, set_mat1, set_mat2, ref_frame_tx_dof1, ref_frame_tx_dof2, is_zshift1, is_zshift2, tx_param_list,
        ghostfrag_surffrag_pair_list, complete_ghost_frag_list, complete_surf_frag_list, log_filepath,
        degen_subdir_out_path, rot_subdir_out_path, ijk_intfrag_cluster_info_dict, result_design_sym, uc_spec_string,
        design_dim, pdb1_path, pdb2_path, expand_matrices, eul_lookup,
        rot_mat1=None, rot_mat2=None, max_z_val=2.0, output_exp_assembly=False, output_uc=False,
        output_surrounding_uc=False, clash_dist=2.2, min_matched=3):

    for i in range(len(tx_param_list)):

        log_file = open(log_filepath, "a+")
        log_file.write("Optimal Shift %s" % str(i) + "\n")
        log_file.close()

        # Dictionaries for PDB1 and PDB2 with (ch_id, res_num) tuples as keys for every residue that is covered by at
        # least 1 matched fragment. Dictionary values are lists containing 1 / (1 + z^2) values for every fragment
match
        # that covers the (ch_id, res_num) residue.
        chid_resnum_scores_dict_pdb1 = {}
        chid_resnum_scores_dict_pdb2 = {}

        # Lists of unique (pdb1/2 chain id, pdb1/2 central residue number) tuples for pdb1/pdb2 interface mono
fragments
        # that were matched to an i,j,k fragment in the database with a z value <= 1.
        # This is to keep track of and to count unique 'high quality' matches.
        unique_interface_monofrags_infolist_highqual_pdb1 = []
        unique_interface_monofrags_infolist_highqual_pdb2 = []

        # Number of unique interface mono fragments matched with a z value <= 1 ('high quality match')
        # This value has to be >= min_matched (minimum number of high quality matches required)
        # for a pose to be selected
        high_qual_match_count = 0

        unique_matched_interface_monofrag_count = 0
        unique_total_interface_monofrags_count = 0
        frag_match_info_list = []
        unique_interface_monofrags_infolist_pdb1 = []
        unique_interface_monofrags_infolist_pdb2 = []
        percent_of_interface_covered = 0.0

        # Keep track of match information and residue pair frequencies for each fragment match
        # this information will be used to calculate a weighted frequency average
        # for all central residues of matched fragments
        res_pair_freq_info_list = []

        tx_parameters = tx_param_list[i][0]
        initial_overlap_z_val = tx_param_list[i][1]
        ghostfrag_surffrag_pair = ghostfrag_surffrag_pair_list[i]

        # Get Optimal External DOF shifts
        n_dof_external = len(get_ext_dof(ref_frame_tx_dof1, ref_frame_tx_dof2))
        optimal_ext_dof_shifts = None
        if n_dof_external > 0:
            optimal_ext_dof_shifts = tx_parameters[0:n_dof_external]

        copy_rot_tr_set_time_start = time.time()
```

**classes/FragDock.py**

```
        # Get Oligomer1 Optimal Internal Translation vector
        representative_int_dof_tx_param_1 = None
        if is_zshift1:
            representative_int_dof_tx_param_1 = [0, 0, tx_parameters[n_dof_external: n_dof_external + 1][0]]

        # Get Oligomer1 Optimal External Translation vector
        representative_ext_dof_tx_params_1 = None
        if optimal_ext_dof_shifts is not None:
            representative_ext_dof_tx_params_1 = get_optimal_external_tx_vector(ref_frame_tx_dof1,
                                                                                  optimal_ext_dof_shifts)

        # Get Oligomer2 Optimal Internal Translation vector
        representative_int_dof_tx_param_2 = None
        if is_zshift2:
            representative_int_dof_tx_param_2 = [0, 0, tx_parameters[n_dof_external + 1: n_dof_external + 2][0]]

        # Get Oligomer2 Optimal External Translation vector
        representative_ext_dof_tx_params_2 = None
        if optimal_ext_dof_shifts is not None:
            representative_ext_dof_tx_params_2 = get_optimal_external_tx_vector(ref_frame_tx_dof2,
                                                                                  optimal_ext_dof_shifts)

        # Get Unit Cell Dimensions for 2D and 3D SCMs
        # Restrict all reference frame translation parameters to > 0 for SCMs with reference frame translational d.o.
f.
        ref_frame_var_is_pos = False
        uc_dimensions = None
        if optimal_ext_dof_shifts is not None:
            ref_frame_tx_dof_e = 0
            ref_frame_tx_dof_f = 0
            ref_frame_tx_dof_g = 0
            if len(optimal_ext_dof_shifts) == 1:
                ref_frame_tx_dof_e = optimal_ext_dof_shifts[0]
                if ref_frame_tx_dof_e > 0:
                    ref_frame_var_is_pos = True
            if len(optimal_ext_dof_shifts) == 2:
                ref_frame_tx_dof_e = optimal_ext_dof_shifts[0]
                ref_frame_tx_dof_f = optimal_ext_dof_shifts[1]
                if ref_frame_tx_dof_e > 0 and ref_frame_tx_dof_f > 0:
                    ref_frame_var_is_pos = True
            if len(optimal_ext_dof_shifts) == 3:
                ref_frame_tx_dof_e = optimal_ext_dof_shifts[0]
                ref_frame_tx_dof_f = optimal_ext_dof_shifts[1]
                ref_frame_tx_dof_g = optimal_ext_dof_shifts[2]
                if ref_frame_tx_dof_e > 0 and ref_frame_tx_dof_f > 0 and ref_frame_tx_dof_g > 0:
                    ref_frame_var_is_pos = True

            uc_dimensions = get_uc_dimensions(uc_spec_string,
                                              ref_frame_tx_dof_e,
                                              ref_frame_tx_dof_f,
                                              ref_frame_tx_dof_g)

        if (optimal_ext_dof_shifts is not None and ref_frame_var_is_pos) or (optimal_ext_dof_shifts is None):

            # Rotate, Translate and Set PDB1
            pdb1_copy = rot_txint_set_txext_pdb(pdb1,
                                                rot_mat=rot_mat1,
                                                internal_tx_vec=representative_int_dof_tx_param_1,
                                                set_mat=set_mat1,
                                                ext_tx_vec=representative_ext_dof_tx_params_1)

            # Rotate, Translate and Set PDB2
            pdb2_copy = rot_txint_set_txext_pdb(pdb2,
                                                rot_mat=rot_mat2,
                                                internal_tx_vec=representative_int_dof_tx_param_2,
                                                set_mat=set_mat2,
                                                ext_tx_vec=representative_ext_dof_tx_params_2)

            copy_rot_tr_set_time_stop = time.time()
            copy_rot_tr_set_time = copy_rot_tr_set_time_stop - copy_rot_tr_set_time_start
            log_file = open(log_filepath, "a+")
            log_file.write("\tCopy and Transform Oligomer1 and Oligomer2 (took: %s s)\n" % str(copy_rot_tr_set_time))
            log_file.close()

            # Check if PDB1 and PDB2 backbones clash
            oligomer1_oligomer2_clash_time_start = time.time()
            kdtree_oligomer1_backbone = sklearn.neighbors.BallTree(np.array(pdb1_copy.extract_backbone_coords()))
            cb_clash_count = kdtree_oligomer1_backbone.two_point_correlation(pdb2_copy.extract_backbone_coords(),
                                                                             [clash_dist])
            oligomer1_oligomer2_clash_time_end = time.time()
```

**classes/FragDock.py**

```
            oligomer1_oligomer2_clash_time = oligomer1_oligomer2_clash_time_end -
oligomer1_oligomer2_clash_time_start

            if cb_clash_count[0] == 0:

                log_file = open(log_filepath, "a+")
                log_file.write("\tNO Backbone Clash when Oligomer1 and Oligomer2 are Docked (took: %s s)"
                               % str(oligomer1_oligomer2_clash_time) + "\n")
                log_file.close()

                # Full Interface Fragment Match
                get_int_ghost_surf_frags_time_start = time.time()
                interface_ghostfrag_list, int_monofrag2_list, interface_ghostfrag_guide_coords_list,
int_monofrag2_guide_coords_list, unique_interface_frag_count_pdb1, unique_interface_frag_count_pdb2 =
get_interface_ghost_surf_frags(pdb1_copy, pdb2_copy, complete_ghost_frag_list, complete_surf_frag_list, rot_mat1,
rot_mat2, representative_int_dof_tx_param_1, representative_int_dof_tx_param_2, set_mat1, set_mat2,
representative_ext_dof_tx_params_1, representative_ext_dof_tx_params_2)
                get_int_ghost_surf_frags_time_end = time.time()
                get_int_ghost_surf_frags_time = get_int_ghost_surf_frags_time_end -
get_int_ghost_surf_frags_time_start

                unique_total_interface_monofrags_count = unique_interface_frag_count_pdb1 +
unique_interface_frag_count_pdb2

                if unique_total_interface_monofrags_count > 0:

                    log_file = open(log_filepath, "a+")
                    log_file.write("\tNewly Formed Interface Contains %s "
                                   "Unique Fragments on Oligomer 1 and %s on Oligomer 2\n"
                                   % (str(unique_interface_frag_count_pdb1), str(unique_interface_frag_count_pdb2)))
                    log_file.write("\t(took: %s s to get interface surface fragments and interface ghost fragments"
                                   " with their guide atoms)\n" % str(get_int_ghost_surf_frags_time))
                    log_file.close()

                    # Get (Oligomer1 Interface Ghost Fragment, Oligomer2 Interface Mono Fragment) guide
                    # coordinate pairs in the same Euler rotational space bucket
                    eul_lookup_start_time = time.time()
                    eul_lookup_all_to_all_list = eul_lookup.check_lookup_table(interface_ghostfrag_guide_coords_list,
                                                                               int_monofrag2_guide_coords_list)
                    eul_lookup_true_list = [(true_tup[0], true_tup[1]) for true_tup in eul_lookup_all_to_all_list if
true_tup[2]]
                    eul_lookup_end_time = time.time()
                    eul_lookup_time = eul_lookup_end_time - eul_lookup_start_time

                    # Get RMSD and z-value for the selected (Ghost Fragment, Interface Fragment) guide coodinate
pairs
                    pair_count = 0
                    total_overlap_count = 0
                    overlap_score_time_start = time.time()
                    for index_pair in eul_lookup_true_list:
                        interface_ghost_frag = interface_ghostfrag_list[index_pair[0]]
                        interface_ghost_frag_guide_coords = interface_ghostfrag_guide_coords_list[index_pair[0]]
                        ghost_frag_i_type = interface_ghost_frag.get_i_frag_type()
                        ghost_frag_j_type = interface_ghost_frag.get_j_frag_type()
                        ghost_frag_k_type = interface_ghost_frag.get_k_frag_type()
                        cluster_id = "i%s_j%s_k%s" % (ghost_frag_i_type, ghost_frag_j_type, ghost_frag_k_type)
                        interface_ghost_frag_cluster_rmsd = ijk_intfrag_cluster_info_dict[ghost_frag_i_type][
ghost_frag_j_type][ghost_frag_k_type].get_rmsd()
                        interface_ghost_frag_cluster_res_freq_list = ijk_intfrag_cluster_info_dict[ghost_frag_i_type]
[ghost_frag_j_type][ghost_frag_k_type].get_central_residue_pair_freqs()

                        interface_mono_frag_guide_coords = int_monofrag2_guide_coords_list[index_pair[1]]
                        interface_mono_frag = int_monofrag2_list[index_pair[1]]
                        interface_mono_frag_type = interface_mono_frag.get_type()

                        if (interface_mono_frag_type == ghost_frag_j_type) and (interface_ghost_frag_cluster_rmsd > 0
):
                            # Calculate RMSD
                            total_overlap_count += 1
                            e1 = euclidean_squared_3d(interface_mono_frag_guide_coords[0],
                                                      interface_ghost_frag_guide_coords[0])
                            e2 = euclidean_squared_3d(interface_mono_frag_guide_coords[1],
                                                      interface_ghost_frag_guide_coords[1])
                            e3 = euclidean_squared_3d(interface_mono_frag_guide_coords[2],
                                                      interface_ghost_frag_guide_coords[2])
                            sum = e1 + e2 + e3
                            mean = sum / float(3)
                            rmsd = math.sqrt(mean)

                            # Calculate Guide Atom Overlap Z-Value
                            # and Calculate Score Term for Nanohedra Residue Level Summation Score
```

**classes/FragDock.py**

```
                            z_val = rmsd / float(interface_ghost_frag_cluster_rmsd)

                            if z_val <= max_z_val:

                                pair_count += 1

                                pdb1_interface_surffrag_ch_id, pdb1_interface_surffrag_central_res_num =
interface_ghost_frag.get_aligned_surf_frag_central_res_tup()
                                pdb2_interface_surffrag_ch_id, pdb2_interface_surffrag_central_res_num =
interface_mono_frag.get_central_res_tup()

                                score_term = 1 / float(1 + (z_val ** 2))

                                covered_residues_pdb1 = [(pdb1_interface_surffrag_ch_id,
pdb1_interface_surffrag_central_res_num + j) for j in range(-2, 3)]
                                covered_residues_pdb2 = [(pdb2_interface_surffrag_ch_id,
pdb2_interface_surffrag_central_res_num + j) for j in range(-2, 3)]
                                for k in range(5):
                                    chid1, resnum1 = covered_residues_pdb1[k]
                                    chid2, resnum2 = covered_residues_pdb2[k]
                                    if (chid1, resnum1) not in chid_resnum_scores_dict_pdb1:
                                        chid_resnum_scores_dict_pdb1[(chid1, resnum1)] = [score_term]
                                    else:
                                        chid_resnum_scores_dict_pdb1[(chid1, resnum1)].append(score_term)

                                    if (chid2, resnum2) not in chid_resnum_scores_dict_pdb2:
                                        chid_resnum_scores_dict_pdb2[(chid2, resnum2)] = [score_term]
                                    else:
                                        chid_resnum_scores_dict_pdb2[(chid2, resnum2)].append(score_term)

                                if z_val <= 1:
                                    if (pdb1_interface_surffrag_ch_id, pdb1_interface_surffrag_central_res_num) not
in unique_interface_monofrags_infolist_highqual_pdb1:
                                        unique_interface_monofrags_infolist_highqual_pdb1.append((
pdb1_interface_surffrag_ch_id, pdb1_interface_surffrag_central_res_num))
                                    if (pdb2_interface_surffrag_ch_id, pdb2_interface_surffrag_central_res_num) not
in unique_interface_monofrags_infolist_highqual_pdb2:
                                        unique_interface_monofrags_infolist_highqual_pdb2.append((
pdb2_interface_surffrag_ch_id, pdb2_interface_surffrag_central_res_num))

                                if (pdb1_interface_surffrag_ch_id, pdb1_interface_surffrag_central_res_num) not in
unique_interface_monofrags_infolist_pdb1:
                                    unique_interface_monofrags_infolist_pdb1.append((pdb1_interface_surffrag_ch_id,
pdb1_interface_surffrag_central_res_num))

                                if (pdb2_interface_surffrag_ch_id, pdb2_interface_surffrag_central_res_num) not in
unique_interface_monofrags_infolist_pdb2:
                                    unique_interface_monofrags_infolist_pdb2.append((pdb2_interface_surffrag_ch_id,
pdb2_interface_surffrag_central_res_num))

                                frag_match_info_list.append((interface_ghost_frag, interface_mono_frag, z_val,
cluster_id,
                                                             pair_count, interface_ghost_frag_cluster_res_freq_list,
                                                             interface_ghost_frag_cluster_rmsd))

                    unique_matched_interface_monofrag_count = len(unique_interface_monofrags_infolist_pdb1) + len(
unique_interface_monofrags_infolist_pdb2)
                    percent_of_interface_covered = unique_matched_interface_monofrag_count / float(
unique_total_interface_monofrags_count)

                    overlap_score_time_stop = time.time()
                    overlap_score_time = overlap_score_time_stop - overlap_score_time_start

                    log_file = open(log_filepath, "a+")
                    log_file.write("\t%s Fragment Match(es) Found in Complete Cluster "
                                   "Representative Fragment Library\n" % str(pair_count))
                    log_file.write("\t(Euler Lookup took %s s for %s fragment pairs and Overlap Score Calculation
took"
                                   " %s for %s fragment pairs)" %
                                   (str(eul_lookup_time), str(len(eul_lookup_all_to_all_list)), str(
overlap_score_time),
                                    str(total_overlap_count)) + "\n")
                    log_file.close()

                    high_qual_match_count = len(unique_interface_monofrags_infolist_highqual_pdb1) + len(
unique_interface_monofrags_infolist_highqual_pdb2)
                    if high_qual_match_count >= min_matched:

                        # Get contacting PDB 1 ASU and PDB 2 ASU
                        asu_pdb_1, asu_pdb_2 = get_contacting_asu(pdb1_copy, pdb2_copy)
```

**classes/FragDock.py**

```
# Check if design has any clashes when expanded
tx_subdir_out_path = rot_subdir_out_path + "/tx_%s" % str(i)
oligomers_subdir = rot_subdir_out_path.split("/")[-3]
degen_subdir = rot_subdir_out_path.split("/")[-2]
rot_subdir = rot_subdir_out_path.split("/")[-1]
pose_id = oligomers_subdir + "_" + degen_subdir + "_" + rot_subdir + "_TX_%s" % str(i)
sampling_id = degen_subdir + "_" + rot_subdir + "_TX_%s" % str(i)
if asu_pdb_1 is not None and asu_pdb_2 is not None:
    exp_des_clash_time_start = time.time()
    exp_des_is_clash = expanded_design_is_clash(asu_pdb_1,
                                                asu_pdb_2,
                                                design_dim,
                                                result_design_sym,
                                                expand_matrices,
                                                uc_dimensions,
                                                tx_subdir_out_path,
                                                output_exp_assembly,
                                                output_uc,
                                                output_surrounding_uc)
    exp_des_clash_time_stop = time.time()
    exp_des_clash_time = exp_des_clash_time_stop - exp_des_clash_time_start

    if not exp_des_is_clash:

        if not os.path.exists(degen_subdir_out_path):
            os.makedirs(degen_subdir_out_path)

        if not os.path.exists(rot_subdir_out_path):
            os.makedirs(rot_subdir_out_path)

        if not os.path.exists(tx_subdir_out_path):
            os.makedirs(tx_subdir_out_path)

        log_file = open(log_filepath, "a+")
        log_file.write("\tNO Backbone Clash when Designed Assembly is Expanded "
                       "(took: %s s including writing)\n" % str(exp_des_clash_time))
        log_file.write("\tSUCCESSFUL DOCKED POSE: %s\n" % tx_subdir_out_path)
        log_file.close()

        # Write PDB1 and PDB2 files
        cryst1_record = None
        if optimal_ext_dof_shifts is not None:
            cryst1_record = generate_cryst1_record(uc_dimensions, result_design_sym)
        pdb1_fname = os.path.splitext(os.path.basename(pdb1.get_filepath()))[0]
        pdb2_fname = os.path.splitext(os.path.basename(pdb2.get_filepath()))[0]
        pdb1_copy.write(tx_subdir_out_path + "/" + pdb1_fname + "_" + sampling_id + ".pdb")
        pdb2_copy.write(tx_subdir_out_path + "/" + pdb2_fname + "_" + sampling_id + ".pdb")

        # Initial Interface Fragment Match
        # Rotate, translate and set initial match interface fragment
        init_match_ghost_frag = ghostfrag_surffrag_pair[0]
        init_match_ghost_frag_pdb = init_match_ghost_frag.get_pdb()
        init_match_ghost_frag_pdb_copy = rot_txint_set_txext_pdb(
            init_match_ghost_frag_pdb, rot_mat=rot_mat1,
            internal_tx_vec=representative_int_dof_tx_param_1,
            set_mat=set_mat1, ext_tx_vec=representative_ext_dof_tx_params_1)

        # Make directories to output matched fragment PDB files
        # initial_match for the initial matched fragment
        # high_qual_match for fragments that were matched with z values <= 1
        # low_qual_match for fragments that were matched with z values > 1
        matched_frag_reps_outdir_path = tx_subdir_out_path + "/matched_fragments"
        if not os.path.exists(matched_frag_reps_outdir_path):
            os.makedirs(matched_frag_reps_outdir_path)

        init_match_outdir_path = matched_frag_reps_outdir_path + "/initial_match"
        if not os.path.exists(init_match_outdir_path):
            os.makedirs(init_match_outdir_path)

        high_qual_matches_outdir_path = matched_frag_reps_outdir_path + "/high_qual_match"
        if not os.path.exists(high_qual_matches_outdir_path):
            os.makedirs(high_qual_matches_outdir_path)

        low_qual_matches_outdir_path = matched_frag_reps_outdir_path + "/low_qual_match"
        if not os.path.exists(low_qual_matches_outdir_path):
            os.makedirs(low_qual_matches_outdir_path)

        # Write out initial match interface fragment
        match_number = 0
        init_match_surf_frag = ghostfrag_surffrag_pair[1]
        init_match_ghost_frag_i_type = init_match_ghost_frag.get_i_frag_type()
```

**classes/FragDock.py**

```
                                init_match_ghost_frag_j_type = init_match_ghost_frag.get_j_frag_type()
                                init_match_ghost_frag_k_type = init_match_ghost_frag.get_k_frag_type()
                                init_match_ghost_frag_cluster_res_freq_list = ijk_intfrag_cluster_info_dict[
init_match_ghost_frag_i_type][init_match_ghost_frag_j_type][init_match_ghost_frag_k_type].
get_central_residue_pair_freqs()
                                init_match_cluster_id = "i%s_j%s_k%s" % (init_match_ghost_frag_i_type,
init_match_ghost_frag_j_type, init_match_ghost_frag_k_type)
                                init_match_ghost_frag_pdb_copy.write(
                                    init_match_outdir_path + "/int_frag_i%s_j%s_k%s_0.pdb"
                                    % (init_match_ghost_frag_i_type,
                                       init_match_ghost_frag_j_type,
                                       init_match_ghost_frag_k_type))
                                init_match_ghost_frag_cluster_rmsd = ijk_intfrag_cluster_info_dict[
init_match_ghost_frag_i_type][init_match_ghost_frag_j_type][init_match_ghost_frag_k_type].get_rmsd()
                                write_frag_match_info_file(init_match_ghost_frag, init_match_surf_frag,
                                                           initial_overlap_z_val, init_match_cluster_id,
                                                           0, init_match_ghost_frag_cluster_res_freq_list,
                                                           init_match_ghost_frag_cluster_rmsd,
                                                           matched_frag_reps_outdir_path,
                                                           pose_id, match_number, is_initial_match=True)

                                # For all matched interface fragments
                                # write out aligned cluster representative fragment
                                # write out associated match information to frag_match_info_file.txt
                                # calculate weighted frequency for central residues
                                # write out weighted frequencies to frag_match_info_file.txt
                                for matched_frag in frag_match_info_list:
                                    match_number += 1
                                    interface_ghost_frag = matched_frag[0]
                                    ghost_frag_i_type = interface_ghost_frag.get_i_frag_type()
                                    ghost_frag_j_type = interface_ghost_frag.get_j_frag_type()
                                    ghost_frag_k_type = interface_ghost_frag.get_k_frag_type()
                                    if matched_frag[2] <= 1:
                                        matched_frag_outdir_path = high_qual_matches_outdir_path
                                    else:
                                        matched_frag_outdir_path = low_qual_matches_outdir_path
                                    interface_ghost_frag.get_pdb().write(
                                        matched_frag_outdir_path + "/int_frag_i%s_j%s_k%s_%s.pdb"
                                        % (ghost_frag_i_type, ghost_frag_j_type, ghost_frag_k_type, str(matched_frag[
4])))

                                    write_frag_match_info_file(matched_frag[0], matched_frag[1], matched_frag[2],
                                                               matched_frag[3], matched_frag[4], matched_frag[5],
                                                               matched_frag[6],
                                                               matched_frag_reps_outdir_path,
                                                               pose_id, match_number)

                                    match_res_pair_freq_list = matched_frag[5]
                                    match_cnt_chid1, match_cnt_resnum1 = matched_frag[0].
get_aligned_surf_frag_central_res_tup()
                                    match_cnt_chid2, match_cnt_resnum2 = matched_frag[1].get_central_res_tup()
                                    match_z_val = matched_frag[2]
                                    match_res_pair_freq_info = FragMatchInfo(match_res_pair_freq_list,
                                                                             match_cnt_chid1,
                                                                             match_cnt_resnum1,
                                                                             match_cnt_chid2,
                                                                             match_cnt_resnum2,
                                                                             match_z_val)
                                    res_pair_freq_info_list.append(match_res_pair_freq_info)

                                weighted_seq_freq_info = SeqFreqInfo(res_pair_freq_info_list)
                                weighted_seq_freq_info.write(matched_frag_reps_outdir_path + "/frag_match_info_file.
txt")

                                # Calculate Nanohedra Residue Level Summation Score
                                res_lev_sum_score = 0
                                for res_scores_list1 in chid_resnum_scores_dict_pdb1.values():
                                    n1 = 1
                                    res_scores_list_sorted1 = sorted(res_scores_list1, reverse=True)
                                    for sc1 in res_scores_list_sorted1:
                                        res_lev_sum_score += sc1 * (1/float(n1))
                                        n1 = n1 * 2
                                for res_scores_list2 in chid_resnum_scores_dict_pdb2.values():
                                    n2 = 1
                                    res_scores_list_sorted2 = sorted(res_scores_list2, reverse=True)
                                    for sc2 in res_scores_list_sorted2:
                                        res_lev_sum_score += sc2 * (1/float(n2))
                                        n2 = n2 * 2

                                # Write Out Docked Pose Info to docked_pose_info_file.txt
                                write_docked_pose_info(tx_subdir_out_path, res_lev_sum_score, high_qual_match_count,
                                                       unique_matched_interface_monofrag_count,
```

**classes/FragDock.py**

```
                                                          unique_total_interface_monofrags_count,
                                                          percent_of_interface_covered, rot_mat1,
                                                          representative_int_dof_tx_param_1, set_mat1,
                                                          representative_ext_dof_tx_params_1, rot_mat2,
                                                          representative_int_dof_tx_param_2, set_mat2,
                                                          representative_ext_dof_tx_params_2, cryst1_record, pdb1_path,
                                                          pdb2_path, pose_id)

                                  else:
                                      log_file = open(log_filepath, "a+")
                                      log_file.write("\tBackbone Clash when Designed Assembly is Expanded "
                                                     "(took: %s s)" % str(exp_des_clash_time) + "\n")
                                      log_file.close()

                              else:
                                  log_file = open(log_filepath, "a+")
                                  log_file.write("\tNO Design ASU Found" + "\n")
                                  log_file.close()

                          else:
                              log_file = open(log_filepath, "a+")
                              log_file.write("\t%s < %s Which is Set as the Minimal Required Amount of High Quality "
                                             "Fragment Matches" %(str(high_qual_match_count), str(min_matched)) + "\n")
                              log_file.close()

                      else:
                          log_file = open(log_filepath, "a+")
                          log_file.write("\tNO Interface Mono Fragments Found" + "\n")
                          log_file.close()

                  else:
                      log_file = open(log_filepath, "a+")
                      log_file.write("\tBackbone Clash when Oligomer1 and Oligomer2 are Docked "
                                     "(took: %s s)" % str(oligomer1_oligomer2_clash_time) + "\n")
                      log_file.close()
              else:
                  efg_tx_params_str = [str(None), str(None), str(None)]
                  for param_index in range(len(optimal_ext_dof_shifts)):
                      efg_tx_params_str[param_index] = str(optimal_ext_dof_shifts[param_index])
                  log_file = open(log_filepath, "a+")
                  log_file.write(
                      "\tReference Frame Shift Parameter(s) is/are Negative: e: %s, f: %s, g: %s\n\n"
                      % (efg_tx_params_str[0], efg_tx_params_str[1], efg_tx_params_str[2]))
                  log_file.close()


def dock(init_intfrag_cluster_rep_dict, ijk_intfrag_cluster_rep_dict, init_monofrag_cluster_rep_pdb_dict_1,
         init_monofrag_cluster_rep_pdb_dict_2, init_intfrag_cluster_info_dict, ijk_monofrag_cluster_rep_pdb_dict,
         ijk_intfrag_cluster_info_dict, free_sasa_exe_path, master_outdir, pdb1_path, pdb2_path, set_mat1, set_mat2,
         ref_frame_tx_dof1, ref_frame_tx_dof2, is_zshift1, is_zshift2, result_design_sym, uc_spec_string, design_dim,
         expand_matrices, eul_lookup, init_max_z_val, subseq_max_z_val, degeneracy_matrices_1=None,
         degeneracy_matrices_2=None, rot_step_deg_pdb1=1, rot_range_deg_pdb1=0, rot_step_deg_pdb2=1,
         rot_range_deg_pdb2=0, output_exp_assembly=False, output_uc=False, output_surrounding_uc=False, min_matched=3
):

    # Output Directory
    pdb1_filename = os.path.splitext(os.path.basename(pdb1_path))[0]
    pdb2_filename = os.path.splitext(os.path.basename(pdb2_path))[0]
    outdir = master_outdir + "/" + pdb1_filename + "_" + pdb2_filename
    if not os.path.exists(outdir):
        os.makedirs(outdir)
    log_filepath = outdir + "/" + pdb1_filename + "_" + pdb2_filename + "_" + "log.txt"

    # Write to Logfile
    log_file = open(log_filepath, "a+")
    log_file.write("DOCKING %s TO %s\n\n" % (pdb1_filename, pdb2_filename))
    log_file.write("Oligomer 1 Path: " + pdb1_path + "\n")
    log_file.write("Oligomer 2 Path: " + pdb2_path + "\n")
    log_file.write("Output Directory: " + outdir + "\n\n")
    log_file.close()

    # Get PDB1 Symmetric Building Block
    pdb1 = PDB()
    pdb1.readfile(pdb1_path)

    # Get Oligomer 1 Ghost Fragments With Guide Coordinates Using Initial Match Fragment Database
    log_file = open(log_filepath, "a+")
    log_file.write("Getting %s Oligomer 1 Ghost Fragments Using INITIAL Fragment Database" % pdb1_filename)
    log_file.close()
    get_init_ghost_frags_time_start = time.time()
    kdtree_oligomer1_backbone = sklearn.neighbors.BallTree(np.array(pdb1.extract_backbone_coords()))
```

**classes/FragDock.py**

```
    surf_frags_1 = get_surface_fragments(pdb1, free_sasa_exe_path)
    ghost_frag_list = []
    ghost_frag_guide_coords_list = []
    for frag1 in surf_frags_1:
        monofrag1 = MonoFragment(frag1, init_monofrag_cluster_rep_pdb_dict_1)
        monofrag_ghostfrag_list = monofrag1.get_ghost_fragments(init_intfrag_cluster_rep_dict,
                                                                kdtree_oligomer1_backbone)
        if monofrag_ghostfrag_list is not None:
            for ghostfrag in monofrag_ghostfrag_list:
                ghost_frag_list.append(ghostfrag)
                ghost_frag_guide_coords_list.append(ghostfrag.get_guide_coords())
    get_init_ghost_frags_time_stop = time.time()
    get_init_ghost_frags_time = get_init_ghost_frags_time_stop - get_init_ghost_frags_time_start
    log_file = open(log_filepath, "a+")
    log_file.write(" (took: %s s)\n" % str(get_init_ghost_frags_time))
    log_file.close()

    # Get Oligomer1 Ghost Fragments With Guide Coordinates Using COMPLETE Fragment Database
    log_file = open(log_filepath, "a+")
    log_file.write("Getting %s Oligomer 1 Ghost Fragments Using COMPLETE Fragment Database" % pdb1_filename)
    log_file.close()
    get_complete_ghost_frags_time_start = time.time()
    complete_ghost_frag_list = []
    for frag1 in surf_frags_1:
        complete_monofrag1 = MonoFragment(frag1, ijk_monofrag_cluster_rep_pdb_dict)
        complete_monofrag1_ghostfrag_list = complete_monofrag1.get_ghost_fragments(
            ijk_intfrag_cluster_rep_dict, kdtree_oligomer1_backbone)
        if complete_monofrag1_ghostfrag_list is not None:
            for complete_ghostfrag in complete_monofrag1_ghostfrag_list:
                complete_ghost_frag_list.append(complete_ghostfrag)
    get_complete_ghost_frags_time_stop = time.time()
    get_complete_ghost_frags_time = get_complete_ghost_frags_time_stop - get_complete_ghost_frags_time_start
    log_file = open(log_filepath, "a+")
    log_file.write(" (took: %s s)\n" % str(get_complete_ghost_frags_time))
    log_file.close()

    # Get PDB2 Symmetric Building Block
    pdb2 = PDB()
    pdb2.readfile(pdb2_path)

    # Get Oligomer 2 Surface (Mono) Fragments With Guide Coordinates Using Initial Match Fragment Database
    get_init_surf_frags_time_start = time.time()
    log_file = open(log_filepath, "a+")
    log_file.write("Getting Oligomer 2 Surface Fragments Using INITIAL Fragment Database")
    log_file.close()
    surf_frags_2 = get_surface_fragments(pdb2, free_sasa_exe_path)
    surf_frag_list = []
    surf_frags_oligomer_2_guide_coords_list = []
    for frag2 in surf_frags_2:
        monofrag2 = MonoFragment(frag2, init_monofrag_cluster_rep_pdb_dict_2)
        monofrag2_guide_coords = monofrag2.get_guide_coords()
        if monofrag2_guide_coords is not None:
            surf_frag_list.append(monofrag2)
            surf_frags_oligomer_2_guide_coords_list.append(monofrag2_guide_coords)
    get_init_surf_frags_time_stop = time.time()
    get_init_surf_frags_time = get_init_surf_frags_time_stop - get_init_surf_frags_time_start
    log_file = open(log_filepath, "a+")
    log_file.write(" (took: %s s)\n" % str(get_init_surf_frags_time))
    log_file.close()

    # Get Oligomer 2 Surface (Mono) Fragments With Guide Coordinates Using COMPLETE Fragment Database
    get_complete_surf_frags_time_start = time.time()
    log_file = open(log_filepath, "a+")
    log_file.write("Getting Oligomer 2 Surface Fragments Using COMPLETE Fragment Database")
    log_file.close()
    complete_surf_frag_list = []
    for frag2 in surf_frags_2:
        complete_monofrag2 = MonoFragment(frag2, ijk_monofrag_cluster_rep_pdb_dict)
        complete_monofrag2_guide_coords = complete_monofrag2.get_guide_coords()
        if complete_monofrag2_guide_coords is not None:
            complete_surf_frag_list.append(complete_monofrag2)
    get_complete_surf_frags_time_stop = time.time()
    get_complete_surf_frags_time = get_complete_surf_frags_time_stop - get_complete_surf_frags_time_start
    log_file = open(log_filepath, "a+")
    log_file.write(" (took: %s s)\n\n" % str(get_complete_surf_frags_time))
    log_file.close()

    # Oligomer 1 Has Interior Rotational Degree of Freedom True or False
    has_int_rot_dof_1 = False
    if rot_range_deg_pdb1 != 0:
        has_int_rot_dof_1 = True
```

**classes/FragDock.py**

```
    # Oligomer 2 Has Interior Rotational Degree of Freedom True or False
    has_int_rot_dof_2 = False
    if rot_range_deg_pdb2 != 0:
        has_int_rot_dof_2 = True

    # Obtain Reference Frame Translation Info
    parsed_ref_frame_tx_dof1 = parse_ref_tx_dof_str_to_list(ref_frame_tx_dof1)
    parsed_ref_frame_tx_dof2 = parse_ref_tx_dof_str_to_list(ref_frame_tx_dof2)

    if parsed_ref_frame_tx_dof1 == ['0', '0', '0'] and parsed_ref_frame_tx_dof2 == ['0', '0', '0']:
        dof_ext = np.empty((0, 3), float)

    else:
        dof_ext = get_ext_dof(ref_frame_tx_dof1, ref_frame_tx_dof2)

    # Transpose Setting Matrices to Set Guide Coordinates Just for Euler Lookup Using np.matmul
    set_mat1_np_t = np.transpose(set_mat1)
    set_mat2_np_t = np.transpose(set_mat2)

    if (degeneracy_matrices_1 is None and has_int_rot_dof_1 is False) and (degeneracy_matrices_2 is None and
has_int_rot_dof_2 is False):

        # No Degeneracies/Rotation Matrices to get for Oligomer1
        rot1_mat = None
        degen1_count = 0
        rot1_count = 0
        log_file = open(log_filepath, "a+")
        log_file.write("No Rotation/Degeneracy Matrices for Oligomer 1" + "\n")

        # No Degeneracies/Rotation Matrices to get for Oligomer2
        rot2_mat = None
        degen2_count = 0
        rot2_count = 0
        log_file.write("No Rotation/Degeneracy Matrices for Oligomer 2\n" + "\n")

        log_file.write("\n***** OLIGOMER 1: Degeneracy %s Rotation %s | OLIGOMER 2: Degeneracy %s Rotation %s *****"
                       % (str(degen1_count), str(rot1_count), str(degen2_count), str(rot2_count)) + "\n")

        # Get (Oligomer1 Ghost Fragment, Oligomer2 Surface Fragment)
        # guide coodinate pairs in the same Euler rotational space bucket
        log_file.write(
            "Get Ghost Fragment/Surface Fragment guide coordinate pairs in the same Euler rotational space bucket\n")
        log_file.close()

        ghost_frag_guide_coords_list_set_for_eul = np.matmul(ghost_frag_guide_coords_list, set_mat1_np_t)
        surf_frags_2_guide_coords_list_set_for_eul = np.matmul(surf_frags_oligomer_2_guide_coords_list, set_mat2_np_t
)

        eul_lookup_all_to_all_list = eul_lookup.check_lookup_table(ghost_frag_guide_coords_list_set_for_eul,
                                                                   surf_frags_2_guide_coords_list_set_for_eul)
        eul_lookup_true_list = [(true_tup[0], true_tup[1]) for true_tup in eul_lookup_all_to_all_list if true_tup[2]]

        # Get optimal shift parameters for the selected (Ghost Fragment, Surface Fragment) guide coodinate pairs
        log_file = open(log_filepath, "a+")
        log_file.write(
            "Get optimal shift parameters for the selected Ghost Fragment/Surface Fragment guide coordinate pairs\n")
        log_file.close()

        ghostfrag_surffrag_pair_list = []
        tx_param_list = []
        for index_pair in eul_lookup_true_list:
            ghost_frag = ghost_frag_list[index_pair[0]]
            ghost_frag_guide_coords = ghost_frag_guide_coords_list[index_pair[0]]
            i_type = ghost_frag.get_i_frag_type()
            j_type = ghost_frag.get_j_frag_type()
            k_type = ghost_frag.get_k_frag_type()
            ghost_frag_cluster_rmsd = init_intfrag_cluster_info_dict[i_type][j_type][k_type].get_rmsd()

            surf_frag_guide_coords = surf_frags_oligomer_2_guide_coords_list[index_pair[1]]
            surf_frag = surf_frag_list[index_pair[1]]
            surf_frag_type = surf_frag.get_type()

            if surf_frag_type == j_type:
                o = OptimalTx(set_mat1, set_mat2, is_zshift1, is_zshift2, ghost_frag_cluster_rmsd,
                              ghost_frag_guide_coords, surf_frag_guide_coords, dof_ext)
                o.apply()

                if o.get_zvalue() <= init_max_z_val:
                    ghostfrag_surffrag_pair_list.append((ghost_frag, surf_frag))
                    # [OptimalExternalDOFShifts, OptimalInternalDOFShifts]
```

**classes/FragDock.py**

```
                    all_optimal_shifts = o.get_all_optimal_shifts()
                    tx_param_list.append((all_optimal_shifts, o.get_zvalue()))

        if len(tx_param_list) == 0:
            log_file = open(log_filepath, "a+")
            log_file.write("No Initial Interface Fragment Matches Found\n\n")
            log_file.close()
        elif len(tx_param_list) == 1:
            log_file = open(log_filepath, "a+")
            log_file.write("1 Initial Interface Fragment Match Found\n")
            log_file.close()
        else:
            log_file = open(log_filepath, "a+")
            log_file.write(
                "%s Initial Interface Fragment Matches Found\n"
                % str(len(tx_param_list)))
            log_file.close()

        degen_subdir_out_path = outdir + "/DEGEN_" + str(degen1_count) + "_" + str(degen2_count)
        rot_subdir_out_path = degen_subdir_out_path + "/ROT_" + str(rot1_count) + "_" + str(rot2_count)

        out(pdb1, pdb2, set_mat1, set_mat2, ref_frame_tx_dof1, ref_frame_tx_dof2, is_zshift1, is_zshift2,
tx_param_list,
            ghostfrag_surffrag_pair_list, complete_ghost_frag_list, complete_surf_frag_list, log_filepath,
            degen_subdir_out_path, rot_subdir_out_path, ijk_intfrag_cluster_info_dict, result_design_sym,
            uc_spec_string, design_dim, pdb1_path, pdb2_path, expand_matrices,
            eul_lookup, rot1_mat, rot2_mat, max_z_val=subseq_max_z_val, output_exp_assembly=output_exp_assembly,
            output_uc=output_uc, output_surrounding_uc=output_surrounding_uc, min_matched=min_matched)

    elif (degeneracy_matrices_1 is not None or has_int_rot_dof_1 is True) and (degeneracy_matrices_2 is None and
has_int_rot_dof_2 is False):
        # Get Degeneracies/Rotation Matrices for Oligomer1: degen_rot_mat_1
        log_file = open(log_filepath, "a+")
        log_file.write("Obtaining Rotation/Degeneracy Matrices for Oligomer 1" + "\n")
        log_file.close()
        rotation_matrices_1 = get_rot_matrices(rot_step_deg_pdb1, "z", rot_range_deg_pdb1)
        degen_rot_mat_1 = get_degen_rotmatrices(degeneracy_matrices_1, rotation_matrices_1)

        # No Degeneracies/Rotation Matrices to get for Oligomer2
        rot2_mat = None
        degen2_count = 0
        rot2_count = 0
        log_file = open(log_filepath, "a+")
        log_file.write("No Rotation/Degeneracy Matrices for Oligomer 2\n" + "\n")
        log_file.close()
        surf_frags_2_guide_coords_list_set_for_eul = np.matmul(surf_frags_oligomer_2_guide_coords_list, set_mat2_np_t
)

        degen1_count = 0
        for degen1 in degen_rot_mat_1:
            degen1_count += 1
            rot1_count = 0
            for rot1_mat in degen1:
                rot1_count += 1

                # Rotate Oligomer1 Ghost Fragment Guide Coodinates using rot1_mat
                rot1_mat_np_t = np.transpose(rot1_mat)
                ghost_frag_guide_coords_list_rot_np = np.matmul(ghost_frag_guide_coords_list, rot1_mat_np_t)
                ghost_frag_guide_coords_list_rot = ghost_frag_guide_coords_list_rot_np.tolist()

                log_file = open(log_filepath, "a+")
                log_file.write(
                    "\n***** OLIGOMER 1: Degeneracy %s Rotation %s | OLIGOMER 2: Degeneracy %s Rotation %s *****"
                    % (str(degen1_count), str(rot1_count), str(degen2_count), str(rot2_count)) + "\n")
                log_file.close()

                # Get (Oligomer1 Ghost Fragment (rotated), Oligomer2 Surface Fragment)
                # guide coodinate pairs in the same Euler rotational space bucket
                log_file = open(log_filepath, "a+")
                log_file.write(
                    "Get Ghost Fragment/Surface Fragment guide coordinate "
                    "pairs in the same Euler rotational space bucket\n")
                log_file.close()

                ghost_frag_guide_coords_list_rot_and_set_for_eul = np.matmul(ghost_frag_guide_coords_list_rot,
                                                                             set_mat1_np_t)

                eul_lookup_all_to_all_list = eul_lookup.check_lookup_table(
                    ghost_frag_guide_coords_list_rot_and_set_for_eul,
                    surf_frags_2_guide_coords_list_set_for_eul)
                eul_lookup_true_list = [(true_tup[0], true_tup[1]) for true_tup in eul_lookup_all_to_all_list if
true_tup[2]]

                # Get optimal shift parameters for the selected (Ghost Fragment, Surface Fragment) guide coodinate
pairs
                log_file = open(log_filepath, "a+")
                log_file.write(
                    "Get optimal shift parameters for the selected "
                    "Ghost Fragment/Surface Fragment guide coordinate pairs\n")
                log_file.close()

                ghostfrag_surffrag_pair_list = []
                tx_param_list = []
                for index_pair in eul_lookup_true_list:
                    ghost_frag = ghost_frag_list[index_pair[0]]
                    ghost_frag_guide_coords = ghost_frag_guide_coords_list_rot[index_pair[0]]
                    i_type = ghost_frag.get_i_frag_type()
                    j_type = ghost_frag.get_j_frag_type()
                    k_type = ghost_frag.get_k_frag_type()
                    ghost_frag_cluster_rmsd = init_intfrag_cluster_info_dict[i_type][j_type][k_type].get_rmsd()

                    surf_frag_guide_coords = surf_frags_oligomer_2_guide_coords_list[index_pair[1]]
                    surf_frag = surf_frag_list[index_pair[1]]
                    surf_frag_type = surf_frag.get_type()

                    if surf_frag_type == j_type:
                        o = OptimalTx(set_mat1, set_mat2, is_zshift1, is_zshift2, ghost_frag_cluster_rmsd,
                                      ghost_frag_guide_coords, surf_frag_guide_coords, dof_ext)
                        o.apply()

                        if o.get_zvalue() <= init_max_z_val:
                            ghostfrag_surffrag_pair_list.append((ghost_frag, surf_frag))
                            # [OptimalExternalDOFShifts, OptimalInternalDOFShifts]
                            all_optimal_shifts = o.get_all_optimal_shifts()
                            tx_param_list.append((all_optimal_shifts, o.get_zvalue()))

                if len(tx_param_list) == 0:
                    log_file = open(log_filepath, "a+")
                    log_file.write(
                        "No Initial Interface Fragment Matches Found\n\n")
                    log_file.close()
                elif len(tx_param_list) == 1:
                    log_file = open(log_filepath, "a+")
                    log_file.write(
                        "1 Initial Interface Fragment Match Found\n")
                    log_file.close()
                else:
                    log_file = open(log_filepath, "a+")
                    log_file.write(
                        "%s Initial Interface Fragment Matches Found" % str(
                            len(tx_param_list)) + "\n")
                    log_file.close()

                degen_subdir_out_path = outdir + "/DEGEN_" + str(degen1_count) + "_" + str(degen2_count)
                rot_subdir_out_path = degen_subdir_out_path + "/ROT_" + str(rot1_count) + "_" + str(rot2_count)

                out(pdb1, pdb2, set_mat1, set_mat2, ref_frame_tx_dof1, ref_frame_tx_dof2, is_zshift1, is_zshift2,
                    tx_param_list, ghostfrag_surffrag_pair_list, complete_ghost_frag_list, complete_surf_frag_list,
                    log_filepath, degen_subdir_out_path, rot_subdir_out_path, ijk_intfrag_cluster_info_dict,
                    result_design_sym, uc_spec_string, design_dim, pdb1_path,
                    pdb2_path, expand_matrices, eul_lookup, rot1_mat, rot2_mat, max_z_val=subseq_max_z_val,
                    output_exp_assembly=output_exp_assembly, output_uc=output_uc,
                    output_surrounding_uc=output_surrounding_uc, min_matched=min_matched)

    elif (degeneracy_matrices_1 is None and has_int_rot_dof_1 is False) and (degeneracy_matrices_2 is not None or
has_int_rot_dof_2 is True):
        # No Degeneracies/Rotation Matrices to get for Oligomer1
        rot1_mat = None
        degen1_count = 0
        rot1_count = 0
        log_file = open(log_filepath, "a+")
        log_file.write("No Rotation/Degeneracy Matrices for Oligomer 1" + "\n")
        log_file.close()
        ghost_frag_guide_coords_list_set_for_eul = np.matmul(ghost_frag_guide_coords_list, set_mat1_np_t)

        # Get Degeneracies/Rotation Matrices for Oligomer2: degen_rot_mat_2
        log_file = open(log_filepath, "a+")
        log_file.write("Obtaining Rotation/Degeneracy Matrices for Oligomer 2\n" + "\n")
        log_file.close()
        rotation_matrices_2 = get_rot_matrices(rot_step_deg_pdb2, "z", rot_range_deg_pdb2)
        degen_rot_mat_2 = get_degen_rotmatrices(degeneracy_matrices_2, rotation_matrices_2)
```

**classes/FragDock.py**

```
        degen2_count = 0
        for degen2 in degen_rot_mat_2:
            degen2_count += 1
            rot2_count = 0
            for rot2_mat in degen2:
                rot2_count += 1

                # Rotate Oligomer2 Surface Fragment Guide Coodinates using rot2_mat
                rot2_mat_np_t = np.transpose(rot2_mat)
                surf_frags_2_guide_coords_list_rot_np = np.matmul(surf_frags_oligomer_2_guide_coords_list,
                                                                  rot2_mat_np_t)
                surf_frags_2_guide_coords_list_rot = surf_frags_2_guide_coords_list_rot_np.tolist()

                log_file = open(log_filepath, "a+")
                log_file.write(
                    "\n***** OLIGOMER 1: Degeneracy %s Rotation %s | OLIGOMER 2: Degeneracy %s Rotation %s *****"
                    % (str(degen1_count), str(rot1_count), str(degen2_count), str(rot2_count)) + "\n")
                log_file.close()

                # Get (Oligomer1 Ghost Fragment, Oligomer2 (rotated) Surface Fragment) guide
                # coodinate pairs in the same Euler rotational space bucket
                log_file = open(log_filepath, "a+")
                log_file.write(
                    "Get Ghost Fragment/Surface Fragment guide coordinate "
                    "pairs in the same Euler rotational space bucket" + "\n")
                log_file.close()

                surf_frags_2_guide_coords_list_rot_and_set_for_eul = np.matmul(surf_frags_2_guide_coords_list_rot,
                                                                               set_mat2_np_t)

                eul_lookup_all_to_all_list = eul_lookup.check_lookup_table(
                    ghost_frag_guide_coords_list_set_for_eul,
                    surf_frags_2_guide_coords_list_rot_and_set_for_eul)
                eul_lookup_true_list = [(true_tup[0], true_tup[1]) for true_tup in eul_lookup_all_to_all_list if
true_tup[2]]

                # Get optimal shift parameters for the selected (Ghost Fragment, Surface Fragment) guide coodinate
pairs
                log_file = open(log_filepath, "a+")
                log_file.write(
                    "Get optimal shift parameters for the selected "
                    "Ghost Fragment/Surface Fragment guide coordinate pairs\n")
                log_file.close()

                ghostfrag_surffrag_pair_list = []
                tx_param_list = []
                for index_pair in eul_lookup_true_list:
                    ghost_frag = ghost_frag_list[index_pair[0]]
                    ghost_frag_guide_coords = ghost_frag_guide_coords_list[index_pair[0]]
                    i_type = ghost_frag.get_i_frag_type()
                    j_type = ghost_frag.get_j_frag_type()
                    k_type = ghost_frag.get_k_frag_type()
                    ghost_frag_cluster_rmsd = init_intfrag_cluster_info_dict[i_type][j_type][k_type].get_rmsd()

                    surf_frag_guide_coords = surf_frags_2_guide_coords_list_rot[index_pair[1]]
                    surf_frag = surf_frag_list[index_pair[1]]
                    surf_frag_type = surf_frag.get_type()

                    if surf_frag_type == j_type:
                        o = OptimalTx(set_mat1, set_mat2, is_zshift1, is_zshift2, ghost_frag_cluster_rmsd,
                                      ghost_frag_guide_coords, surf_frag_guide_coords, dof_ext)
                        o.apply()

                        if o.get_zvalue() <= init_max_z_val:
                            ghostfrag_surffrag_pair_list.append((ghost_frag, surf_frag))
                            # [OptimalExternalDOFShifts, OptimalInternalDOFShifts]
                            all_optimal_shifts = o.get_all_optimal_shifts()
                            tx_param_list.append((all_optimal_shifts, o.get_zvalue()))

                if len(tx_param_list) == 0:
                    log_file = open(log_filepath, "a+")
                    log_file.write("No Initial Interface Fragment Matches Found\n\n")
                    log_file.close()
                elif len(tx_param_list) == 1:
                    log_file = open(log_filepath, "a+")
                    log_file.write("1 Initial Interface Fragment Match Found\n")
                    log_file.close()
                else:
                    log_file = open(log_filepath, "a+")
                    log_file.write("%s Initial Interface Fragment Matches Found\n"
                                   % str(len(tx_param_list)))
```

**classes/FragDock.py**

```
                    log_file.close()

                degen_subdir_out_path = outdir + "/DEGEN_" + str(degen1_count) + "_" + str(degen2_count)
                rot_subdir_out_path = degen_subdir_out_path + "/ROT_" + str(rot1_count) + "_" + str(rot2_count)

                out(pdb1, pdb2, set_mat1, set_mat2, ref_frame_tx_dof1, ref_frame_tx_dof2, is_zshift1, is_zshift2,
                    tx_param_list, ghostfrag_surffrag_pair_list, complete_ghost_frag_list, complete_surf_frag_list,
                    log_filepath, degen_subdir_out_path, rot_subdir_out_path, ijk_intfrag_cluster_info_dict,
                    result_design_sym, uc_spec_string, design_dim, pdb1_path,
                    pdb2_path, expand_matrices, eul_lookup, rot1_mat, rot2_mat, max_z_val=subseq_max_z_val,
                    output_exp_assembly=output_exp_assembly, output_uc=output_uc,
                    output_surrounding_uc=output_surrounding_uc, min_matched=min_matched)

    elif (degeneracy_matrices_1 is not None or has_int_rot_dof_1 is True) and (degeneracy_matrices_2 is not None or
has_int_rot_dof_2 is True):

        log_file = open(log_filepath, "a+")
        log_file.write("Obtaining Rotation/Degeneracy Matrices for Oligomer 1" + "\n")
        log_file.close()

        # Get Degeneracies/Rotation Matrices for Oligomer1: degen_rot_mat_1
        rotation_matrices_1 = get_rot_matrices(rot_step_deg_pdb1, "z", rot_range_deg_pdb1)
        degen_rot_mat_1 = get_degen_rotmatrices(degeneracy_matrices_1, rotation_matrices_1)

        log_file = open(log_filepath, "a+")
        log_file.write("Obtaining Rotation/Degeneracy Matrices for Oligomer 2\n" + "\n")
        log_file.close()
        # Get Degeneracies/Rotation Matrices for Oligomer2: degen_rot_mat_2
        rotation_matrices_2 = get_rot_matrices(rot_step_deg_pdb2, "z", rot_range_deg_pdb2)
        degen_rot_mat_2 = get_degen_rotmatrices(degeneracy_matrices_2, rotation_matrices_2)

        degen1_count = 0
        for degen1 in degen_rot_mat_1:
            degen1_count += 1

            rot1_count = 0
            for rot1_mat in degen1:
                rot1_count += 1

                # Rotate Oligomer1 Ghost Fragment Guide Coordinates using rot1_mat
                rot1_mat_np_t = np.transpose(rot1_mat)
                ghost_frag_guide_coords_list_rot_np = np.matmul(ghost_frag_guide_coords_list, rot1_mat_np_t)
                ghost_frag_guide_coords_list_rot = ghost_frag_guide_coords_list_rot_np.tolist()

                ghost_frag_guide_coords_list_rot_and_set_for_eul = np.matmul(ghost_frag_guide_coords_list_rot,
                                                                             set_mat1_np_t)

                degen2_count = 0
                for degen2 in degen_rot_mat_2:
                    degen2_count += 1

                    rot2_count = 0
                    for rot2_mat in degen2:
                        rot2_count += 1

                        # Rotate Oligomer2 Surface Fragment Guide Coordinates using rot2_mat
                        rot2_mat_np_t = np.transpose(rot2_mat)
                        surf_frags_2_guide_coords_list_rot_np = np.matmul(surf_frags_oligomer_2_guide_coords_list,
                                                                          rot2_mat_np_t)
                        surf_frags_2_guide_coords_list_rot = surf_frags_2_guide_coords_list_rot_np.tolist()

                        log_file = open(log_filepath, "a+")
                        log_file.write(
                            "\n***** OLIGOMER 1: Degeneracy %s Rotation %s "
                            "| OLIGOMER 2: Degeneracy %s Rotation %s *****"
                            % (str(degen1_count), str(rot1_count), str(degen2_count), str(rot2_count)) + "\n")
                        log_file.close()

                        # Get (Oligomer1 Ghost Fragment (rotated), Oligomer2 (rotated) Surface Fragment) guide
                        # coodinate pairs in the same Euler rotational space bucket
                        log_file = open(log_filepath, "a+")
                        log_file.write(
                            "Get Ghost Fragment/Surface Fragment guide coordinate pairs "
                            "in the same Euler rotational space bucket\n")
                        log_file.close()

                        eul_time_start = time.time()
                        surf_frags_2_guide_coords_list_rot_and_set_for_eul = np.matmul(
                            surf_frags_2_guide_coords_list_rot, set_mat2_np_t)

                        eul_lookup_all_to_all_list = eul_lookup.check_lookup_table(
                            ghost_frag_guide_coords_list_rot_and_set_for_eul,
                            surf_frags_2_guide_coords_list_rot_and_set_for_eul)
                        eul_lookup_true_list = [(true_tup[0], true_tup[1]) for true_tup in eul_lookup_all_to_all_list
 if true_tup[2]]

                        eul_time_stop = time.time()
                        eul_time = eul_time_stop - eul_time_start

                        # Euler TIME TEST
                        log_file = open(log_filepath, "a+")
                        log_file.write("Euler Search Took: %s s for %s ghost/surf pairs\n"
                                       % (str(eul_time), str(len(eul_lookup_all_to_all_list))))
                        log_file.close()

                        # Get optimal shift parameters for the selected (Ghost Fragment, Surface Fragment)
                        # guide coodinate pairs
                        log_file = open(log_filepath, "a+")
                        log_file.write(
                            "Get optimal shift parameters for the selected "
                            "Ghost Fragment/Surface Fragment guide coordinate pairs\n")
                        log_file.close()

                        ghostfrag_surffrag_pair_list = []
                        tx_param_list = []
                        opt_tx_time_start = time.time()
                        opt_tx_count = 0
                        for index_pair in eul_lookup_true_list:
                            ghost_frag = ghost_frag_list[index_pair[0]]
                            ghost_frag_guide_coords = ghost_frag_guide_coords_list_rot[index_pair[0]]
                            i_type = ghost_frag.get_i_frag_type()
                            j_type = ghost_frag.get_j_frag_type()
                            k_type = ghost_frag.get_k_frag_type()
                            ghost_frag_cluster_rmsd = init_intfrag_cluster_info_dict[i_type][j_type][k_type].get_rmsd
()

                            surf_frag_guide_coords = surf_frags_2_guide_coords_list_rot[index_pair[1]]
                            surf_frag = surf_frag_list[index_pair[1]]
                            surf_frag_type = surf_frag.get_type()

                            if surf_frag_type == j_type:
                                opt_tx_count += 1
                                o = OptimalTx(set_mat1, set_mat2, is_zshift1, is_zshift2, ghost_frag_cluster_rmsd,
                                              ghost_frag_guide_coords, surf_frag_guide_coords, dof_ext)
                                o.apply()

                                if o.get_zvalue() <= init_max_z_val:
                                    ghostfrag_surffrag_pair_list.append((ghost_frag, surf_frag))
                                    # [OptimalExternalDOFShifts, OptimalInternalDOFShifts]
                                    all_optimal_shifts = o.get_all_optimal_shifts()
                                    tx_param_list.append((all_optimal_shifts, o.get_zvalue()))

                        # Optimal Shift Time Test
                        opt_tx_time_stop = time.time()
                        opt_tx_time = opt_tx_time_stop - opt_tx_time_start
                        log_file = open(log_filepath, "a+")
                        log_file.write("Optimal Shift Search Took: %s s for %s guide coordinate pairs\n"
                                       % (str(opt_tx_time), str(opt_tx_count)))
                        log_file.close()

                        if len(tx_param_list) == 0:
                            log_file = open(log_filepath, "a+")
                            log_file.write(
                                "No Initial Interface Fragment Matches Found\n\n")
                            log_file.close()
                        elif len(tx_param_list) == 1:
                            log_file = open(log_filepath, "a+")
                            log_file.write(
                                "1 Initial Interface Fragment Match Found\n")
                            log_file.close()
                        else:
                            log_file = open(log_filepath, "a+")
                            log_file.write(
                                "%s Initial Interface Fragment Matches Found\n"
                                % str(len(tx_param_list)))
                            log_file.close()

                        degen_subdir_out_path = outdir + "/DEGEN_" + str(degen1_count) + "_" + str(degen2_count)
                        rot_subdir_out_path = degen_subdir_out_path + "/ROT_" + str(rot1_count) + "_" + str(
rot2_count)

                        out(pdb1, pdb2, set_mat1, set_mat2, ref_frame_tx_dof1, ref_frame_tx_dof2, is_zshift1,
                            is_zshift2, tx_param_list, ghostfrag_surffrag_pair_list, complete_ghost_frag_list,
```

**classes/FragDock.py**

```
                                  complete_surf_frag_list, log_filepath, degen_subdir_out_path, rot_subdir_out_path,
                                  ijk_intfrag_cluster_info_dict, result_design_sym, uc_spec_string, design_dim,
                                  pdb1_path, pdb2_path, expand_matrices, eul_lookup,
                                  rot1_mat, rot2_mat, max_z_val=subseq_max_z_val, output_exp_assembly=output_exp_assembly,
                                  output_uc=output_uc, output_surrounding_uc=output_surrounding_uc, min_matched=min_matched
 )
```

**classes/FragDock.py**

WeightedSeqFreq.py

```
# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"



class FragMatchInfo:
    def __init__(self, res_pair_freqs, oligomer_1_ch_id, oligomer_1_res_num, oligomer_2_ch_id, oligomer_2_res_num,
z_val):
        self.res_pair_freqs = res_pair_freqs
        self.oligomer_1_ch_id = oligomer_1_ch_id
        self.oligomer_1_res_num = oligomer_1_res_num
        self.oligomer_2_ch_id = oligomer_2_ch_id
        self.oligomer_2_res_num = oligomer_2_res_num
        self.z_val = z_val

        self.score = 1 / float(1 + (z_val ** 2))

        self.oligomer_1_res_freqs = {}
        self.oligomer_2_res_freqs = {}
        for res_pair in self.res_pair_freqs:
            res1_type, res2_type = res_pair[0]
            res_freq = res_pair[1]

            if res1_type in self.oligomer_1_res_freqs:
                self.oligomer_1_res_freqs[res1_type] += res_freq
            else:
                self.oligomer_1_res_freqs[res1_type] = res_freq

            if res2_type in self.oligomer_2_res_freqs:
                self.oligomer_2_res_freqs[res2_type] += res_freq
            else:
                self.oligomer_2_res_freqs[res2_type] = res_freq

    def get_res_pair_freqs(self):
        return self.res_pair_freqs

    def get_oligomer_1_ch_id(self):
        return self.oligomer_1_ch_id

    def get_oligomer_1_res_num(self):
        return self.oligomer_1_res_num

    def get_oligomer_2_ch_id(self):
        return self.oligomer_2_ch_id

    def get_oligomer_2_res_num(self):
        return self.oligomer_2_res_num

    def get_z_val(self):
        return self.z_val

    def get_oligomer_1_res_freqs(self):
        return self.oligomer_1_res_freqs

    def get_oligomer_2_res_freqs(self):
        return self.oligomer_2_res_freqs

    def get_score(self):
        return self.score


class SeqFreqInfo:
    def __init__(self, frag_match_info_list):
        self.frag_match_info_list = frag_match_info_list

        self.oligomer_1 = []
        self.oligomer_2 = []

        oligomer_1_freqs_w_sum = {}
        oligomer_2_freqs_w_sum = {}
        score_sum_dict_1 = {}
        score_sum_dict_2 = {}
        for frag_match_info in frag_match_info_list:
            # get information from specific match
            match_score = frag_match_info.get_score()
            match_oligomer_1_ch_id = frag_match_info.get_oligomer_1_ch_id()
            match_oligomer_1_res_num = frag_match_info.get_oligomer_1_res_num()
            match_oligomer_1_freqs = frag_match_info.get_oligomer_1_res_freqs()
            match_oligomer_2_ch_id = frag_match_info.get_oligomer_2_ch_id()
```

**classes/WeightedSeqFreq.py**

```
            match_oligomer_2_res_num = frag_match_info.get_oligomer_2_res_num()
            match_oligomer_2_freqs = frag_match_info.get_oligomer_2_res_freqs()

            # weigh matched residue frequencies by match score
            # do so for the matched residue on oligomer 1 and the matched residue on oligomer 2
            match_oligomer_1_freqs_w = {res_type: freq*match_score for (res_type, freq) in match_oligomer_1_freqs.
items()}
            match_oligomer_2_freqs_w = {res_type: freq*match_score for (res_type, freq) in match_oligomer_2_freqs.
items()}

            # add match score to sum of the residue match scores
            # do so for the matched residue on oligomer 1 and the matched residue on oligomer 2
            if match_oligomer_1_ch_id in score_sum_dict_1:
                if match_oligomer_1_res_num in score_sum_dict_1[match_oligomer_1_ch_id]:
                    score_sum_dict_1[match_oligomer_1_ch_id][match_oligomer_1_res_num] += match_score
                else:
                    score_sum_dict_1[match_oligomer_1_ch_id][match_oligomer_1_res_num] = match_score
            else:
                score_sum_dict_1[match_oligomer_1_ch_id] = {match_oligomer_1_res_num: match_score}

            if match_oligomer_2_ch_id in score_sum_dict_2:
                if match_oligomer_2_res_num in score_sum_dict_2[match_oligomer_2_ch_id]:
                    score_sum_dict_2[match_oligomer_2_ch_id][match_oligomer_2_res_num] += match_score
                else:
                    score_sum_dict_2[match_oligomer_2_ch_id][match_oligomer_2_res_num] = match_score
            else:
                score_sum_dict_2[match_oligomer_2_ch_id] = {match_oligomer_2_res_num: match_score}

            # for each residue type add the weighted residue frequency to the sum of weighted residue type
frequencies
            # do so for the matched residue on oligomer 1 and the matched residue on oligomer 2
            if match_oligomer_1_ch_id in oligomer_1_freqs_w_sum:
                if match_oligomer_1_res_num in oligomer_1_freqs_w_sum[match_oligomer_1_ch_id]:
                    for (match_res_type, match_freq) in match_oligomer_1_freqs_w.items():
                        if match_res_type in oligomer_1_freqs_w_sum[match_oligomer_1_ch_id][match_oligomer_1_res_num]
:
                            oligomer_1_freqs_w_sum[match_oligomer_1_ch_id][match_oligomer_1_res_num][match_res_type]
+= match_freq
                        else:
                            oligomer_1_freqs_w_sum[match_oligomer_1_ch_id][match_oligomer_1_res_num][match_res_type]
= match_freq
                else:
                    oligomer_1_freqs_w_sum[match_oligomer_1_ch_id][match_oligomer_1_res_num] =
match_oligomer_1_freqs_w
            else:
                oligomer_1_freqs_w_sum[match_oligomer_1_ch_id] = {match_oligomer_1_res_num: match_oligomer_1_freqs_w}

            if match_oligomer_2_ch_id in oligomer_2_freqs_w_sum:
                if match_oligomer_2_res_num in oligomer_2_freqs_w_sum[match_oligomer_2_ch_id]:
                    for (match_res_type, match_freq) in match_oligomer_2_freqs_w.items():
                        if match_res_type in oligomer_2_freqs_w_sum[match_oligomer_2_ch_id][match_oligomer_2_res_num]
:
                            oligomer_2_freqs_w_sum[match_oligomer_2_ch_id][match_oligomer_2_res_num][match_res_type]
+= match_freq
                        else:
                            oligomer_2_freqs_w_sum[match_oligomer_2_ch_id][match_oligomer_2_res_num][match_res_type]
= match_freq
                else:
                    oligomer_2_freqs_w_sum[match_oligomer_2_ch_id][match_oligomer_2_res_num] =
match_oligomer_2_freqs_w
            else:
                oligomer_2_freqs_w_sum[match_oligomer_2_ch_id] = {match_oligomer_2_res_num: match_oligomer_2_freqs_w}

        # for each residue calculate the weighted average frequency for the different residue types
        for ch_id_1 in oligomer_1_freqs_w_sum:
            for res_num_1 in oligomer_1_freqs_w_sum[ch_id_1]:
                res_score_sum_1 = score_sum_dict_1[ch_id_1][res_num_1]
                for res_type_1 in oligomer_1_freqs_w_sum[ch_id_1][res_num_1]:
                    oligomer_1_freqs_w_sum[ch_id_1][res_num_1][res_type_1] /= float(res_score_sum_1)

        for ch_id_2 in oligomer_2_freqs_w_sum:
            for res_num_2 in oligomer_2_freqs_w_sum[ch_id_2]:
                res_score_sum_2 = score_sum_dict_2[ch_id_2][res_num_2]
                for res_type_2 in oligomer_2_freqs_w_sum[ch_id_2][res_num_2]:
                    oligomer_2_freqs_w_sum[ch_id_2][res_num_2][res_type_2] /= float(res_score_sum_2)

        # sort sequence frequency information by chain ID, residue number and frequency
        # for oligomer 1 and oligomer 2
        for (ch_id_1, ch_id_1_resnums) in sorted(oligomer_1_freqs_w_sum.items(), key=lambda kv: kv[0]):
            sorted_freqs_1 = []
            for (res_num_1, res_num_1_freqs) in sorted(oligomer_1_freqs_w_sum[ch_id_1].items(), key=lambda kv: kv[0])
```

**classes/WeightedSeqFreq.py**

```
:
                sorted_freqs_1.append((res_num_1, sorted(oligomer_1_freqs_w_sum[ch_id_1][res_num_1].items(), key=
lambda kv: kv[1], reverse=True)))
            self.oligomer_1.append((ch_id_1, sorted_freqs_1))

        for (ch_id_2, ch_id_2_resnums) in sorted(oligomer_2_freqs_w_sum.items(), key=lambda kv: kv[0]):
            sorted_freqs_2 = []
            for (res_num_2, res_num_2_freqs) in sorted(oligomer_2_freqs_w_sum[ch_id_2].items(), key=lambda kv: kv[0])
:
                sorted_freqs_2.append((res_num_2, sorted(oligomer_2_freqs_w_sum[ch_id_2][res_num_2].items(), key=
lambda kv: kv[1], reverse=True)))
            self.oligomer_2.append((ch_id_2, sorted_freqs_2))

    def get_oligomer_1(self):
        return self.oligomer_1

    def get_oligomer_2(self):
        return self.oligomer_2

    def write(self, output_file_path):

        outfile = open(output_file_path, "a+")
        outfile.write("***** WEIGHTED RESIDUE FREQUENCIES *****\n\n")
        outfile.close()

        oligomers_seq_freqs = (self.get_oligomer_1(), self.get_oligomer_2())

        # write output for oligomers 1 and 2 to output file
        for oligomer in range(2):
            for (ch_id, ch_id_resnums) in oligomers_seq_freqs[oligomer]:
                outfile = open(output_file_path, "a+")
                outfile.write("OLIGOMER %s  |  CHAIN %s\n" % (str(oligomer + 1), ch_id))
                ch_id_res_freqs = []
                for (res_num, res_num_freqs) in ch_id_resnums:
                    res_num_freqs_rounded = [(res_type, round(res_freq, 3)) for (res_type, res_freq) in res_num_freqs
]
                    outfile.write("RES NUM " + str(res_num) + "\n" + str(res_num_freqs_rounded) + "\n\n")
                    ch_id_res_freqs.append(res_num_freqs)
                outfile.write("\n")
                outfile.close()
```

**utils**

SamplingUtils.py

```python
import numpy as np
import math


# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"


# ROTATION RANGE DEG
C2 = 180
C3 = 120
C4 = 90
C5 = 72
C6 = 60
RotRangeDict = {"C2": C2, "C3": C3, "C4": C4, "C5": C5, "C6": C6}


def get_degeneracy_matrices(oligomer_symmetry_1, oligomer_symmetry_2, design_dimension, design_symmetry):
    valid_pt_gp_symm_list = ["C2", "C3", "C4", "C5", "C6", "D2", "D3", "D4", "D6", "T", "O", "I"]

    if oligomer_symmetry_1 not in valid_pt_gp_symm_list:
        raise ValueError("Invalid Point Group Symmetry")

    if oligomer_symmetry_2 not in valid_pt_gp_symm_list:
        raise ValueError("Invalid Point Group Symmetry")

    if design_symmetry not in valid_pt_gp_symm_list:
        raise ValueError("Invalid Point Group Symmetry")

    if design_dimension not in [0, 2, 3]:
        raise ValueError("Invalid Design Dimension")

    degeneracies = [None, None]

    for i in range(2):

        degeneracy_matrices = None

        oligomer_symmetry = oligomer_symmetry_1 if i == 0 else oligomer_symmetry_2

        # For cages, only one of the two oligomers need to be flipped. By convention we flip oligomer 2.
        if design_dimension == 0 and i == 1:
            degeneracy_matrices = [[[-1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, -1.0]]]  # ROT180y

        # For layers that obey a cyclic point group symmetry
        # and that are constructed from two oligomers that both obey cyclic symmetry
        # only one of the two oligomers need to be flipped. By convention we flip oligomer 2.
        elif design_dimension == 2 and i == 1 and (oligomer_symmetry_1[0], oligomer_symmetry_2[0], design_symmetry[0]
) == ("C", "C", "C"):
            degeneracy_matrices = [[[-1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, -1.0]]]  # ROT180y

        elif oligomer_symmetry in ["D3", "D4", "D6"] and design_symmetry in ["D3", "D4", "D6", "T", "O"]:
            if oligomer_symmetry == "D3":
                degeneracy_matrices = [[[0.5, -0.86603, 0.0], [0.86603, 0.5, 0.0], [0.0, 0.0, 1.0]]]  # ROT60z
            elif oligomer_symmetry == "D4":
                # 45 degrees about z; z unaffected; x goes to [1,-1,0] direction
                degeneracy_matrices = [[[0.707107, 0.707107, 0.0], [-0.707107, 0.707107, 0.0], [0.0, 0.0, 1.0]]]
            elif oligomer_symmetry == "D6":
                degeneracy_matrices = [[[0.86603, -0.5, 0.0], [0.5, 0.86603, 0.0], [0.0, 0.0, 1.0]]]  # ROT30z

        elif oligomer_symmetry == "D2" and design_symmetry != "O":
            if design_symmetry == "T":
                degeneracy_matrices = [[[0.0, -1.0, 0.0], [1.0, 0.0, 0.0], [0.0, 0.0, 1.0]]]  # ROT90z

            elif design_symmetry == "D4":
                degeneracy_matrices = [[[0.0, 0.0, 1.0], [1.0, 0.0, 0.0], [0.0, 1.0, 0.0]],
                                       [[0.0, 1.0, 0.0], [0.0,  0.0, 1.0], [1.0, 0.0, 0.0]]]  # z,x,y and y,z,x

            elif design_symmetry == "D2" or design_symmetry == "D6":
                degeneracy_matrices = [[[0.0, 0.0, 1.0], [1.0, 0.0, 0.0], [0.0, 1.0, 0.0]],
                                       [[0.0, 1.0, 0.0], [0.0,  0.0, 1.0], [1.0, 0.0, 0.0]],
                                       [[-1.0, 0.0, 0.0], [0.0, 0.0, 1.0], [0.0, 1.0, 0.0]],
                                       [[0.0, 0.0, 1.0], [0.0, -1.0, 0.0], [1.0, 0.0, 0.0]],
                                       [[0.0, 1.0, 0.0], [1.0, 0.0, 0.0], [0.0, 0.0, -1.0]]]

        elif oligomer_symmetry == "T" and design_symmetry == "T":
            degeneracy_matrices = [[[0.0, -1.0, 0.0], [1.0, 0.0, 0.0], [0.0, 0.0, 1.0]]]  # ROT90z

        degeneracies[i] = degeneracy_matrices
```

**utils/SamplingUtils.py**

```
    return degeneracies


def parse_ref_tx_dof_str_to_list(ref_frame_tx_dof_string):
    s1 = ref_frame_tx_dof_string.replace('<', '')
    s2 = s1.replace('>', '')
    l1 = s2.split(',')
    l2 = [x.replace(' ', '') for x in l1]
    return l2


def get_tx_dof_ref_frame_var_vec(string_vec, var):
    return_vec = [0.0, 0.0, 0.0]
    for i in range(3):
        if var in string_vec[i] and '*' in string_vec[i]:
            return_vec[i] = float(string_vec[i].split('*')[0])
        elif "-" + var in string_vec[i]:
            return_vec[i] = -1.0
        elif var == string_vec[i]:
            return_vec[i] = 1.0
    return return_vec


def get_ext_dof(ref_frame_tx_dof1, ref_frame_tx_dof2):

    ext_dof = []

    parsed_1 = parse_ref_tx_dof_str_to_list(ref_frame_tx_dof1)
    parsed_2 = parse_ref_tx_dof_str_to_list(ref_frame_tx_dof2)

    e1_var_vec = get_tx_dof_ref_frame_var_vec(parsed_1, 'e')
    f1_var_vec = get_tx_dof_ref_frame_var_vec(parsed_1, 'f')
    g1_var_vec = get_tx_dof_ref_frame_var_vec(parsed_1, 'g')

    e2_var_vec = get_tx_dof_ref_frame_var_vec(parsed_2, 'e')
    f2_var_vec = get_tx_dof_ref_frame_var_vec(parsed_2, 'f')
    g2_var_vec = get_tx_dof_ref_frame_var_vec(parsed_2, 'g')

    e2e1_diff = (np.array(e2_var_vec) - np.array(e1_var_vec)).tolist()
    f2f1_diff = (np.array(f2_var_vec) - np.array(f1_var_vec)).tolist()
    g2g1_diff = (np.array(g2_var_vec) - np.array(g1_var_vec)).tolist()

    if e2e1_diff != [0, 0, 0]:
        ext_dof.append(e2e1_diff)

    if f2f1_diff != [0, 0, 0]:
        ext_dof.append(f2f1_diff)

    if g2g1_diff != [0, 0, 0]:
        ext_dof.append(g2g1_diff)

    return ext_dof


def get_optimal_external_tx_vector(ref_frame_tx_dof, optimal_ext_dof_shifts):

    ext_dof_variables = ['e', 'f', 'g']

    parsed_ref_tx_vec = parse_ref_tx_dof_str_to_list(ref_frame_tx_dof)

    optimal_external_tx_vector = np.array([0.0, 0.0, 0.0])
    for dof_shift_index in range(len(optimal_ext_dof_shifts)):
        dof_shift = optimal_ext_dof_shifts[dof_shift_index]
        var_vec = get_tx_dof_ref_frame_var_vec(parsed_ref_tx_vec, ext_dof_variables[dof_shift_index])
        shifted_var_vec = np.array(var_vec) * dof_shift
        optimal_external_tx_vector += shifted_var_vec

    return optimal_external_tx_vector.tolist()


def get_rot_matrices(step_deg, axis, rot_range_deg):
    rot_matrices = []
    if axis == 'x':
        for angle_deg in range(0, rot_range_deg, step_deg):
            rad = math.radians(float(angle_deg))
            rotmatrix = [[1, 0, 0], [0, math.cos(rad), -1 * math.sin(rad)], [0, math.sin(rad), math.cos(rad)]]
            rot_matrices.append(rotmatrix)
        return rot_matrices

    elif axis == 'y':
```

**utils/SamplingUtils.py**

```
        for angle_deg in range(0, rot_range_deg, step_deg):
            rad = math.radians(float(angle_deg))
            rotmatrix = [[math.cos(rad), 0, math.sin(rad)], [0, 1, 0], [-1 * math.sin(rad), 0, math.cos(rad)]]
            rot_matrices.append(rotmatrix)
        return rot_matrices

    elif axis == 'z':
        for angle_deg in range(0, rot_range_deg, step_deg):
            rad = math.radians(float(angle_deg))
            rotmatrix = [[math.cos(rad), -1 * math.sin(rad), 0], [math.sin(rad), math.cos(rad), 0], [0, 0, 1]]
            rot_matrices.append(rotmatrix)
        return rot_matrices

    else:
        print "AXIS SELECTED FOR SAMPLING IS NOT SUPPORTED"
        return None


def get_degen_rotmatrices(degeneracy_matrices, rotation_matrices):
    if rotation_matrices == list() and degeneracy_matrices is not None:
        identity_matrix = [[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]
        return [[identity_matrix]] + [[degen_mat] for degen_mat in degeneracy_matrices]

    elif rotation_matrices != list() and degeneracy_matrices is None:
        return [rotation_matrices]

    elif rotation_matrices != list() and degeneracy_matrices is not None:
        degen_rotmatrices = [rotation_matrices]
        for degen in degeneracy_matrices:
            degen_list = []
            for rot in rotation_matrices:
                combined = np.matmul(rot, degen)
                degen_list.append(combined.tolist())
            degen_rotmatrices.append(degen_list)
        return degen_rotmatrices
    else:
        identity_matrix = [[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]
        return [[identity_matrix]]
```

**utils/SamplingUtils.py**

ExpandAssemblyUtils.py

```python
import os
import numpy as np
import copy
import pickle
import sys
from classes.PDB import PDB
from classes.Atom import Atom
from utils.GeneralUtils import center_of_mass_3d
import sklearn.neighbors


# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"


def generate_cryst1_record(dimensions, spacegroup):
    # dimensions is a python list containing a, b, c (Angstroms) alpha, beta, gamma (degrees)
    sg_cryst1_fmt_dict = {'F222': 'F 2 2 2', 'P6222': 'P 62 2 2', 'I4132': 'I 41 3 2', 'P432': 'P 4 3 2', 'P6322': 'P
 63 2 2', 'I4122': 'I 41 2 2', 'I213': 'I 21 3', 'I422': 'I 4 2 2', 'I432': 'I 4 3 2', 'P4222': 'P 42 2 2', 'F23': 'F
 2 3', 'P23': 'P 2 3', 'P213': 'P 21 3', 'F432': 'F 4 3 2', 'P622': 'P 6 2 2', 'P4232': 'P 42 3 2', 'F4132': 'F 41 3
2', 'P4132': 'P 41 3 2', 'P422': 'P 4 2 2', 'P312': 'P 3 1 2', 'R32': 'R 3 2'}
    pg_cryst1_fmt_dict = {'p3': 'P 3', 'p321': 'P 3 2 1', 'p622': 'P 6 2 2', 'p4': 'P 4', 'p222': 'P 2 2 2', 'p422':
'P 4 2 2', 'p4212': 'P 4 21 2', 'p6': 'P 6', 'p312': 'P 3 1 2', 'c222': 'C 2 2 2'}
    zvalue_dict = {'P 2 3': 12, 'P 42 2 2': 8, 'P 3 2 1': 6, 'P 63 2 2': 12, 'P 3 1 2': 12, 'P 6 2 2': 12, 'F 2 3':
48, 'F 2 2 2': 16, 'P 62 2 2': 12, 'I 4 2 2': 16, 'I 21 3': 24, 'R 3 2': 6, 'P 4 21 2': 8, 'I 4 3 2': 48, 'P 41 3 2':
 24, 'I 41 3 2': 48, 'P 3': 3, 'P 6': 6, 'I 41 2 2': 16, 'P 4': 4, 'C 2 2 2': 8, 'P 2 2 2': 4, 'P 21 3': 12, 'F 41 3
2': 96, 'P 4 2 2': 8, 'P 4 3 2': 24, 'F 4 3 2': 96, 'P 42 3 2': 24}

    if spacegroup in sg_cryst1_fmt_dict:
        fmt_spacegroup = sg_cryst1_fmt_dict[spacegroup]
        zvalue = zvalue_dict[fmt_spacegroup]
    elif spacegroup in pg_cryst1_fmt_dict:
        fmt_spacegroup = pg_cryst1_fmt_dict[spacegroup]
        zvalue = zvalue_dict[fmt_spacegroup]
        dimensions[2] = 1.0
        dimensions[3] = 90.0
        dimensions[4] = 90.0
    else:
        raise ValueError("SPACEGROUP NOT SUPPORTED")

    cryst1_fmt = "CRYST1{box[0]:9.3f}{box[1]:9.3f}{box[2]:9.3f}""{ang[0]:7.2f}{ang[1]:7.2f}{ang[2]:7.2f} ""{
spacegroup:<11s}{zvalue:4d}\n"
    return cryst1_fmt.format(box=dimensions[:3], ang=dimensions[3:], spacegroup=fmt_spacegroup, zvalue=zvalue)


def get_ptgrp_sym_op(sym_type, expand_matrix_dir=os.path.dirname(os.path.dirname(os.path.realpath(__file__))) + "/
symmetry_operators/POINT_GROUP_SYMM_OPERATORS"):
    expand_matrix_filepath = expand_matrix_dir + "/" + sym_type + ".txt"
    expand_matrix_file = open(expand_matrix_filepath, "r")
    expand_matrix_lines = expand_matrix_file.readlines()
    expand_matrix_file.close()
    line_count = 0
    expand_matrices = []
    mat = []
    for line in expand_matrix_lines:
        line = line.split()
        if len(line) == 3:
            line_float = [float(s) for s in line]
            mat.append(line_float)
            line_count += 1
            if line_count % 3 == 0:
                expand_matrices.append(mat)
                mat = []
    return expand_matrices


def get_expanded_ptgrp_pdbs(pdb1_asu, pdb2_asu, expand_matrices):
    asu_symm_mates = []

    pdb_asu = PDB()
    pdb_asu.set_all_atoms(pdb1_asu.get_all_atoms() + pdb2_asu.get_all_atoms())

    asu_coords = pdb_asu.extract_all_coords()

    for r in expand_matrices:

        r_mat = np.transpose(np.array(r))
        r_asu_coords = np.matmul(asu_coords, r_mat)
```

**utils/ExpandAssemblyUtils.py**

```
        asu_sym_mate_pdb = PDB()
        asu_sym_mate_pdb_atom_list = []
        atom_count = 0
        for atom in pdb_asu.get_all_atoms():
            x_transformed = r_asu_coords[atom_count][0]
            y_transformed = r_asu_coords[atom_count][1]
            z_transformed = r_asu_coords[atom_count][2]
            atom_transformed = Atom(atom_count, atom.get_type(), atom.get_alt_location(),
                                 atom.get_residue_type(), atom.get_chain(),
                                 atom.get_residue_number(),
                                 atom.get_code_for_insertion(), x_transformed, y_transformed,
                                 z_transformed,
                                 atom.get_occ(), atom.get_temp_fact(), atom.get_element_symbol(),
                                 atom.get_atom_charge())
            atom_count += 1
            asu_sym_mate_pdb_atom_list.append(atom_transformed)

        asu_sym_mate_pdb.set_all_atoms(asu_sym_mate_pdb_atom_list)
        asu_symm_mates.append(asu_sym_mate_pdb)

    return asu_symm_mates


def write_expanded_ptgrp(expanded_ptgrp_pdbs, outfile_path):
    outfile = open(outfile_path, "w")
    model_count = 1
    for pdb in expanded_ptgrp_pdbs:
        outfile.write("MODEL     {:>4s}\n".format(str(model_count)))
        model_count += 1
        for atom in pdb.all_atoms:
            outfile.write(str(atom))
        outfile.write("ENDMDL\n")
    outfile.close()


def expanded_ptgrp_is_clash(expanded_ptgrp_pdbs, clash_distance=2.2):

    asu = expanded_ptgrp_pdbs[0]
    symm_mates_wo_asu = expanded_ptgrp_pdbs[1:]

    asu_bb_coords = asu.extract_backbone_coords()
    symm_mates_wo_asu_bb_coords = []
    for sym_mate_pdb in symm_mates_wo_asu:
        symm_mates_wo_asu_bb_coords.extend(sym_mate_pdb.extract_backbone_coords())

    kdtree_central_asu_bb = sklearn.neighbors.BallTree(np.array(asu_bb_coords))
    cb_clash_count = kdtree_central_asu_bb.two_point_correlation(symm_mates_wo_asu_bb_coords, [clash_distance])

    if cb_clash_count[0] == 0:
        return False  # "NO CLASH"

    else:
        return True  # "CLASH!!"


def get_sg_sym_op(sym_type, space_group_operator_dir=os.path.dirname(os.path.dirname(os.path.realpath(__file__))) +
"/symmetry_operators/SPACE_GROUP_SYMM_OPERATORS"):
    sg_op_filepath = space_group_operator_dir + "/" + sym_type + ".pickle"
    sg_op_file = open(sg_op_filepath, "r")
    sg_sym_op = pickle.load(sg_op_file)
    sg_op_file.close()

    return sg_sym_op


def cart_to_frac(cart_coords, dimensions):
    # http://www.ruppweb.org/Xray/tutorial/Coordinate%20system%20transformation.htm
    if len(dimensions) == 6:
        a2r = np.pi / 180.0
        a, b, c, alpha, beta, gamma = dimensions
        alpha *= a2r
        beta *= a2r
        gamma *= a2r

        # volume
        v = a*b*c*np.sqrt((1 - np.cos(alpha)**2 - np.cos(beta)**2 - np.cos(gamma)**2 + 2*(np.cos(alpha) * np.cos(beta
) *np.cos(gamma))))

        # deorthogonalization matrix M
        M_0 = [1/a, -(np.cos(gamma)/float(a*np.sin(gamma))), (((b*np.cos(gamma)*c*(np.cos(alpha)-(np.cos(beta)*np.cos
(gamma)))) / float(np.sin(gamma)))-(b*c*np.cos(beta)*np.sin(gamma)))*(1/float(v))]
```

**utils/ExpandAssemblyUtils.py**

```
        M_1 = [0, 1/(b*np.sin(gamma)), -((a*c*(np.cos(alpha)-(np.cos(beta)*np.cos(gamma))))/float(v*np.sin(gamma)))]
        M_2 = [0, 0,  (a*b*np.sin(gamma))/float(v)]
        M = np.array([M_0, M_1, M_2])

        frac_coords = np.matmul(np.array(cart_coords), np.transpose(M))

        return frac_coords

    else:
        raise ValueError("UNIT CELL DIMENSIONS INCORRECTLY SPECIFIED. CORRECT FORMAT IS: [a, b, c,  alpha, beta,
gamma]")


def frac_to_cart(frac_coords, dimensions):
    # http://www.ruppweb.org/Xray/tutorial/Coordinate%20system%20transformation.htm
    if len(dimensions) == 6:
        a2r = np.pi / 180.0
        a, b, c, alpha, beta, gamma = dimensions
        alpha *= a2r
        beta *= a2r
        gamma *= a2r

        # volume
        v = a*b*c*np.sqrt((1 - np.cos(alpha) ** 2 - np.cos(beta) ** 2 - np.cos(gamma) ** 2 + 2 * (np.cos(alpha) * np.
cos(beta) * np.cos(gamma))))

        # orthogonalization matrix M_inv
        M_inv_0 = [a, b * np.cos(gamma), c * np.cos(beta)]
        M_inv_1 = [0, b * np.sin(gamma), (c * (np.cos(alpha) - (np.cos(beta) * np.cos(gamma)))) / float(np.sin(gamma)
)]
        M_inv_2 = [0, 0, v / float(a * b * np.sin(gamma))]
        M_inv = np.array([M_inv_0, M_inv_1, M_inv_2])

        cart_coords = np.matmul(np.array(frac_coords), np.transpose(M_inv))

        return cart_coords

    else:
        raise ValueError("UNIT CELL DIMENSIONS INCORRECTLY SPECIFIED. CORRECT FORMAT IS: [a, b, c,  alpha, beta,
gamma]")


def get_central_asu_pdb_2d(pdb1, pdb2, uc_dimensions):
    pdb_asu = PDB()
    pdb_asu.read_atom_list(pdb1.get_all_atoms() + pdb2.get_all_atoms())

    pdb_asu_coords_cart = pdb_asu.extract_all_coords()

    asu_com_cart = center_of_mass_3d(pdb_asu_coords_cart)
    asu_com_frac = cart_to_frac(asu_com_cart, uc_dimensions)

    asu_com_x_min_cart = sys.maxint
    x_min_shift_vec_frac = None
    for x in range(-10, 11):
        asu_com_x_shifted_coords_frac = asu_com_frac + [x, 0, 0]
        asu_com_x_shifted_coords_cart = frac_to_cart(asu_com_x_shifted_coords_frac, uc_dimensions)
        if abs(asu_com_x_shifted_coords_cart[0]) < abs(asu_com_x_min_cart):
            asu_com_x_min_cart = asu_com_x_shifted_coords_cart[0]
            x_min_shift_vec_frac = [x, 0, 0]

    asu_com_y_min_cart = sys.maxint
    y_min_shift_vec_frac = None
    for y in range(-10, 11):
        asu_com_y_shifted_coords_frac = asu_com_frac + [0, y, 0]
        asu_com_y_shifted_coords_cart = frac_to_cart(asu_com_y_shifted_coords_frac, uc_dimensions)
        if abs(asu_com_y_shifted_coords_cart[1]) < abs(asu_com_y_min_cart):
            asu_com_y_min_cart = asu_com_y_shifted_coords_cart[1]
            y_min_shift_vec_frac = [0, y, 0]

    if x_min_shift_vec_frac is not None and y_min_shift_vec_frac is not None:
        xyz_min_shift_vec_frac = [x_min_shift_vec_frac[0], y_min_shift_vec_frac[1], 0]

        if xyz_min_shift_vec_frac == [0, 0, 0]:
            return pdb_asu

        else:
            pdb_asu_coords_frac = cart_to_frac(pdb_asu_coords_cart, uc_dimensions)
            xyz_min_shifted_pdb_asu_coords_frac = pdb_asu_coords_frac + xyz_min_shift_vec_frac
            xyz_min_shifted_pdb_asu_coords_cart = frac_to_cart(xyz_min_shifted_pdb_asu_coords_frac, uc_dimensions)

            xyz_min_shifted_asu_pdb = copy.deepcopy(pdb_asu)
```

**utils/ExpandAssemblyUtils.py**

```
            xyz_min_shifted_asu_pdb.replace_coords(xyz_min_shifted_pdb_asu_coords_cart)

            return xyz_min_shifted_asu_pdb

    else:
        return pdb_asu


def get_central_asu_pdb_3d(pdb1, pdb2, uc_dimensions):
    pdb_asu = PDB()
    pdb_asu.read_atom_list(pdb1.get_all_atoms() + pdb2.get_all_atoms())

    pdb_asu_coords_cart = pdb_asu.extract_all_coords()

    asu_com_cart = center_of_mass_3d(pdb_asu_coords_cart)
    asu_com_frac = cart_to_frac(asu_com_cart, uc_dimensions)

    asu_com_x_min_cart = sys.maxint
    x_min_shift_vec_frac = None
    for x in range(-10, 11):
        asu_com_x_shifted_coords_frac = asu_com_frac + [x, 0, 0]
        asu_com_x_shifted_coords_cart = frac_to_cart(asu_com_x_shifted_coords_frac, uc_dimensions)
        if abs(asu_com_x_shifted_coords_cart[0]) < abs(asu_com_x_min_cart):
            asu_com_x_min_cart = asu_com_x_shifted_coords_cart[0]
            x_min_shift_vec_frac = [x, 0, 0]

    asu_com_y_min_cart = sys.maxint
    y_min_shift_vec_frac = None
    for y in range(-10, 11):
        asu_com_y_shifted_coords_frac = asu_com_frac + [0, y, 0]
        asu_com_y_shifted_coords_cart = frac_to_cart(asu_com_y_shifted_coords_frac, uc_dimensions)
        if abs(asu_com_y_shifted_coords_cart[1]) < abs(asu_com_y_min_cart):
            asu_com_y_min_cart = asu_com_y_shifted_coords_cart[1]
            y_min_shift_vec_frac = [0, y, 0]

    asu_com_z_min_cart = sys.maxint
    z_min_shift_vec_frac = None
    for z in range(-10, 11):
        asu_com_z_shifted_coords_frac = asu_com_frac + [0, 0, z]
        asu_com_z_shifted_coords_cart = frac_to_cart(asu_com_z_shifted_coords_frac, uc_dimensions)
        if abs(asu_com_z_shifted_coords_cart[2]) < abs(asu_com_z_min_cart):
            asu_com_z_min_cart = asu_com_z_shifted_coords_cart[2]
            z_min_shift_vec_frac = [0, 0, z]

    if x_min_shift_vec_frac is not None and y_min_shift_vec_frac is not None and z_min_shift_vec_frac is not None:
        xyz_min_shift_vec_frac = [x_min_shift_vec_frac[0], y_min_shift_vec_frac[1], z_min_shift_vec_frac[2]]

        if xyz_min_shift_vec_frac == [0, 0, 0]:
            return pdb_asu

        else:
            pdb_asu_coords_frac = cart_to_frac(pdb_asu_coords_cart, uc_dimensions)
            xyz_min_shifted_pdb_asu_coords_frac = pdb_asu_coords_frac + xyz_min_shift_vec_frac
            xyz_min_shifted_pdb_asu_coords_cart = frac_to_cart(xyz_min_shifted_pdb_asu_coords_frac, uc_dimensions)

            xyz_min_shifted_asu_pdb = copy.deepcopy(pdb_asu)
            xyz_min_shifted_asu_pdb.replace_coords(xyz_min_shifted_pdb_asu_coords_cart)

            return xyz_min_shifted_asu_pdb

    else:
        return pdb_asu


def get_unit_cell_sym_mates(pdb_asu, expand_matrices, uc_dimensions):
    unit_cell_sym_mates = [pdb_asu]

    asu_cart_coords = pdb_asu.extract_all_coords()
    asu_frac_coords = cart_to_frac(asu_cart_coords, uc_dimensions)

    for r, t in expand_matrices:
        t_vec = np.array(t)
        r_mat = np.transpose(np.array(r))

        r_asu_frac_coords = np.matmul(asu_frac_coords, r_mat)
        tr_asu_frac_coords = r_asu_frac_coords + t_vec

        tr_asu_cart_coords = frac_to_cart(tr_asu_frac_coords, uc_dimensions).tolist()

        unit_cell_sym_mate_pdb = PDB()
        unit_cell_sym_mate_pdb_atom_list = []
```

**utils/ExpandAssemblyUtils.py**

```
        atom_count = 0
        for atom in pdb_asu.get_all_atoms():
            x_transformed = tr_asu_cart_coords[atom_count][0]
            y_transformed = tr_asu_cart_coords[atom_count][1]
            z_transformed = tr_asu_cart_coords[atom_count][2]
            atom_transformed = Atom(atom_count, atom.get_type(), atom.get_alt_location(),
                                    atom.get_residue_type(), atom.get_chain(),
                                    atom.get_residue_number(),
                                    atom.get_code_for_insertion(), x_transformed, y_transformed,
                                    z_transformed,
                                    atom.get_occ(), atom.get_temp_fact(), atom.get_element_symbol(),
                                    atom.get_atom_charge())
            atom_count += 1
            unit_cell_sym_mate_pdb_atom_list.append(atom_transformed)

        unit_cell_sym_mate_pdb.set_all_atoms(unit_cell_sym_mate_pdb_atom_list)
        unit_cell_sym_mates.append(unit_cell_sym_mate_pdb)

    return unit_cell_sym_mates


def get_surrounding_unit_cells_2d(unit_cell_sym_mates, uc_dimensions):
    all_surrounding_unit_cells = []

    asu_bb_atom_template = unit_cell_sym_mates[0].get_backbone_atoms()
    unit_cell_sym_mates_len = len(unit_cell_sym_mates)

    central_uc_bb_cart_coords = []
    for unit_cell_sym_mate_pdb in unit_cell_sym_mates:
        central_uc_bb_cart_coords.extend(unit_cell_sym_mate_pdb.extract_backbone_coords())
    central_uc_bb_frac_coords = cart_to_frac(central_uc_bb_cart_coords, uc_dimensions)

    all_surrounding_uc_bb_frac_coords = []
    for x_shift in [-1, 0, 1]:
        for y_shift in [-1, 0, 1]:
            if [x_shift, y_shift] != [0, 0]:
                shifted_uc_bb_frac_coords = central_uc_bb_frac_coords + [x_shift, y_shift, 0]
                all_surrounding_uc_bb_frac_coords.extend(shifted_uc_bb_frac_coords)

    all_surrounding_uc_bb_cart_coords = frac_to_cart(all_surrounding_uc_bb_frac_coords, uc_dimensions)
    all_surrounding_uc_bb_cart_coords = np.split(all_surrounding_uc_bb_cart_coords, 8)

    for surrounding_uc_bb_cart_coords in all_surrounding_uc_bb_cart_coords:
        all_uc_sym_mates_bb_cart_coords = np.split(surrounding_uc_bb_cart_coords, unit_cell_sym_mates_len)
        one_surrounding_unit_cell = []
        for uc_sym_mate_bb_cart_coords in all_uc_sym_mates_bb_cart_coords:
            uc_sym_mate_bb_pdb = PDB()
            uc_sym_mate_bb_atoms = []
            atom_count = 0
            for atom in asu_bb_atom_template:
                x_transformed = uc_sym_mate_bb_cart_coords[atom_count][0]
                y_transformed = uc_sym_mate_bb_cart_coords[atom_count][1]
                z_transformed = uc_sym_mate_bb_cart_coords[atom_count][2]
                atom_transformed = Atom(atom.get_number(), atom.get_type(), atom.get_alt_location(),
                                        atom.get_residue_type(), atom.get_chain(),
                                        atom.get_residue_number(),
                                        atom.get_code_for_insertion(), x_transformed, y_transformed,
                                        z_transformed,
                                        atom.get_occ(), atom.get_temp_fact(), atom.get_element_symbol(),
                                        atom.get_atom_charge())
                uc_sym_mate_bb_atoms.append(atom_transformed)
                atom_count += 1

            uc_sym_mate_bb_pdb.set_all_atoms(uc_sym_mate_bb_atoms)
            one_surrounding_unit_cell.append(uc_sym_mate_bb_pdb)

        all_surrounding_unit_cells.append(one_surrounding_unit_cell)

    return all_surrounding_unit_cells


def get_surrounding_unit_cells_3d(unit_cell_sym_mates, uc_dimensions):
    all_surrounding_unit_cells = []

    asu_bb_atom_template = unit_cell_sym_mates[0].get_backbone_atoms()
    unit_cell_sym_mates_len = len(unit_cell_sym_mates)

    central_uc_bb_cart_coords = []
    for unit_cell_sym_mate_pdb in unit_cell_sym_mates:
        central_uc_bb_cart_coords.extend(unit_cell_sym_mate_pdb.extract_backbone_coords())
    central_uc_bb_frac_coords = cart_to_frac(central_uc_bb_cart_coords, uc_dimensions)
```

**utils/ExpandAssemblyUtils.py**

```
    all_surrounding_uc_bb_frac_coords = []
    for x_shift in [-1, 0, 1]:
        for y_shift in [-1, 0, 1]:
            for z_shift in [-1, 0, 1]:
                if [x_shift, y_shift, z_shift] != [0, 0, 0]:
                    shifted_uc_bb_frac_coords = central_uc_bb_frac_coords + [x_shift, y_shift, z_shift]
                    all_surrounding_uc_bb_frac_coords.extend(shifted_uc_bb_frac_coords)

    all_surrounding_uc_bb_cart_coords = frac_to_cart(all_surrounding_uc_bb_frac_coords, uc_dimensions)
    all_surrounding_uc_bb_cart_coords = np.split(all_surrounding_uc_bb_cart_coords, 26)

    for surrounding_uc_bb_cart_coords in all_surrounding_uc_bb_cart_coords:
        all_uc_sym_mates_bb_cart_coords = np.split(surrounding_uc_bb_cart_coords, unit_cell_sym_mates_len)
        one_surrounding_unit_cell = []
        for uc_sym_mate_bb_cart_coords in all_uc_sym_mates_bb_cart_coords:
            uc_sym_mate_bb_pdb = PDB()
            uc_sym_mate_bb_atoms = []
            atom_count = 0
            for atom in asu_bb_atom_template:
                x_transformed = uc_sym_mate_bb_cart_coords[atom_count][0]
                y_transformed = uc_sym_mate_bb_cart_coords[atom_count][1]
                z_transformed = uc_sym_mate_bb_cart_coords[atom_count][2]
                atom_transformed = Atom(atom.get_number(), atom.get_type(), atom.get_alt_location(),
                                        atom.get_residue_type(), atom.get_chain(),
                                        atom.get_residue_number(),
                                        atom.get_code_for_insertion(), x_transformed, y_transformed,
                                        z_transformed,
                                        atom.get_occ(), atom.get_temp_fact(), atom.get_element_symbol(),
                                        atom.get_atom_charge())
                uc_sym_mate_bb_atoms.append(atom_transformed)
                atom_count += 1

            uc_sym_mate_bb_pdb.set_all_atoms(uc_sym_mate_bb_atoms)
            one_surrounding_unit_cell.append(uc_sym_mate_bb_pdb)

        all_surrounding_unit_cells.append(one_surrounding_unit_cell)

    return all_surrounding_unit_cells


def write_unit_cell_sym_mates(unit_cell_sym_mates, outfile_path):
    f = open(outfile_path, "a+")
    model_count = 0
    for unit_cell_sym_mate_pdb in unit_cell_sym_mates:
        model_count += 1
        model_line = "MODEL     {:>4s}\n".format(str(model_count))
        end_model_line = "ENDMDL\n"

        f.write(model_line)
        for atom in unit_cell_sym_mate_pdb.get_all_atoms():
            f.write(str(atom))
        f.write(end_model_line)
    f.close()


def write_surrounding_unit_cells(surrounding_unit_cells, outfile_path):
    f = open(outfile_path, "a+")

    model_count = 0
    for unit_cell in surrounding_unit_cells:
        for unit_cell_sym_mate_pdb in unit_cell:
            model_count += 1
            model_line = "MODEL     {:>4s}\n".format(str(model_count))
            end_model_line = "ENDMDL\n"

            f.write(model_line)
            for atom in unit_cell_sym_mate_pdb.get_all_atoms():
                f.write(str(atom))
            f.write(end_model_line)

    f.close()


def uc_expansion_is_clash(central_unit_cell, clash_distance=2.2):
    central_asu_pdb = central_unit_cell[0]
    central_unit_cell_wo_central_asu = central_unit_cell[1:]

    central_asu_pdb_bb_coords = central_asu_pdb.extract_backbone_coords()
    central_unit_cell_wo_central_asu_bb_coords = []
    for unit_cell_sym_mate_pdb in central_unit_cell_wo_central_asu:
```

**utils/ExpandAssemblyUtils.py**

```
        central_unit_cell_wo_central_asu_bb_coords.extend(unit_cell_sym_mate_pdb.extract_backbone_coords())

    kdtree_central_asu_bb = sklearn.neighbors.BallTree(np.array(central_asu_pdb_bb_coords))
    cb_clash_count = kdtree_central_asu_bb.two_point_correlation(central_unit_cell_wo_central_asu_bb_coords, [
clash_distance])

    if cb_clash_count[0] == 0:
        return False  # "NO CLASH"

    else:
        return True  # "CLASH!!"


def surrounding_uc_is_clash(central_unit_cell, surrounding_unit_cells, clash_distance=2.2):
    central_asu_pdb = central_unit_cell[0]
    all_unit_cells_wo_central_asu = surrounding_unit_cells + [central_unit_cell[1:]]

    central_asu_pdb_bb_coords = central_asu_pdb.extract_backbone_coords()
    all_unit_cells_wo_central_asu_bb_coords = []
    for unit_cell in all_unit_cells_wo_central_asu:
        for unit_cell_sym_mate_pdb in unit_cell:
            all_unit_cells_wo_central_asu_bb_coords.extend(unit_cell_sym_mate_pdb.extract_backbone_coords())

    kdtree_central_asu_bb = sklearn.neighbors.BallTree(np.array(central_asu_pdb_bb_coords))
    cb_clash_count = kdtree_central_asu_bb.two_point_correlation(all_unit_cells_wo_central_asu_bb_coords, [
clash_distance])

    if cb_clash_count[0] == 0:
        return False  # "NO CLASH"

    else:
        return True  # "CLASH!!"


def expanded_design_is_clash(asu_pdb_1, asu_pdb_2, design_dim, result_design_sym, expand_matrices, uc_dimensions=None
,
                             outdir=None, output_exp_assembly=False, output_uc=False, output_surrounding_uc=False):

    if design_dim == 0:
        expanded_ptgrp_pdbs = get_expanded_ptgrp_pdbs(asu_pdb_1, asu_pdb_2, expand_matrices)

        is_clash = expanded_ptgrp_is_clash(expanded_ptgrp_pdbs)

        if not is_clash and outdir is not None:
            if not os.path.exists(outdir):
                os.makedirs(outdir)
            if output_exp_assembly:
                write_expanded_ptgrp(expanded_ptgrp_pdbs, outdir + "/expanded_assembly.pdb")
            pdb_asu = expanded_ptgrp_pdbs[0]
            pdb_asu.write(outdir + "/asu.pdb")

        return is_clash

    elif design_dim == 2:
        pdb_asu = get_central_asu_pdb_2d(asu_pdb_1, asu_pdb_2, uc_dimensions)

        cryst1_record = generate_cryst1_record(uc_dimensions, result_design_sym)

        unit_cell_pdbs = get_unit_cell_sym_mates(pdb_asu, expand_matrices, uc_dimensions)

        is_uc_exp_clash = uc_expansion_is_clash(unit_cell_pdbs)
        if is_uc_exp_clash:
            return is_uc_exp_clash

        all_surrounding_unit_cells = get_surrounding_unit_cells_2d(unit_cell_pdbs, uc_dimensions)
        is_clash = surrounding_uc_is_clash(unit_cell_pdbs, all_surrounding_unit_cells)

        if not is_clash and outdir is not None:
            if not os.path.exists(outdir):
                os.makedirs(outdir)
            if output_uc:
                write_unit_cell_sym_mates(unit_cell_pdbs, outdir + "/central_uc.pdb")
            if output_surrounding_uc:
                write_surrounding_unit_cells(all_surrounding_unit_cells, outdir + "/surrounding_unit_cells.pdb")
            pdb_asu.write(outdir + "/central_asu.pdb", cryst1=cryst1_record)

        return is_clash

    elif design_dim == 3:

        cryst1_record = generate_cryst1_record(uc_dimensions, result_design_sym)
```

**utils/ExpandAssemblyUtils.py**

```
        pdb_asu = get_central_asu_pdb_3d(asu_pdb_1, asu_pdb_2, uc_dimensions)

        unit_cell_pdbs = get_unit_cell_sym_mates(pdb_asu, expand_matrices, uc_dimensions)

        is_uc_exp_clash = uc_expansion_is_clash(unit_cell_pdbs)

        if is_uc_exp_clash:
            return is_uc_exp_clash

        all_surrounding_unit_cells = get_surrounding_unit_cells_3d(unit_cell_pdbs, uc_dimensions)

        is_clash = surrounding_uc_is_clash(unit_cell_pdbs, all_surrounding_unit_cells)

        if not is_clash and outdir is not None:
            if not os.path.exists(outdir):
                os.makedirs(outdir)
            if output_uc:
                write_unit_cell_sym_mates(unit_cell_pdbs, outdir + "/central_uc.pdb")
            if output_surrounding_uc:
                write_surrounding_unit_cells(all_surrounding_unit_cells, outdir + "/surrounding_unit_cells.pdb")
            pdb_asu.write(outdir + "/central_asu.pdb", cryst1=cryst1_record)

        return is_clash

    else:
        raise ValueError("%s is an Invalid Design Dimension. The Only Valid Dimensions are: 0, 2, 3\n" %str(
design_dim))
```

**utils/ExpandAssemblyUtils.py**

PDBUtils.py

```python
import sklearn.neighbors
from classes.PDB import *
import numpy as np
from classes.Fragment import GhostFragment
from classes.Fragment import MonoFragment
from utils.GeneralUtils import rot_txint_set_txext_frag_coord_sets


# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"


def rot_txint_set_txext_pdb(pdb, rot_mat=None, internal_tx_vec=None, set_mat=None, ext_tx_vec=None):
    pdb_coords = np.array(pdb.extract_all_coords())

    if pdb_coords.size != 0:

        # Rotate coordinates if rotation matrix is provided
        if rot_mat is not None:
            rot_mat_T = np.transpose(rot_mat)
            pdb_coords = np.matmul(pdb_coords, rot_mat_T)

        # Translate coordinates if internal translation vector is provided
        if internal_tx_vec is not None:
            pdb_coords = pdb_coords + internal_tx_vec

        # Set coordinates if setting matrix is provided
        if set_mat is not None:
            set_mat_T = np.transpose(set_mat)
            pdb_coords = np.matmul(pdb_coords, set_mat_T)

        # Translate coordinates if external translation vector is provided
        if ext_tx_vec is not None:
            pdb_coords = pdb_coords + ext_tx_vec

        transformed_pdb = PDB()
        transformed_atoms = []
        atom_index = 0
        for atom in pdb.get_all_atoms():
            x_transformed = pdb_coords[atom_index][0]
            y_transformed = pdb_coords[atom_index][1]
            z_transformed = pdb_coords[atom_index][2]
            atom_transformed = Atom(atom.get_number(), atom.get_type(), atom.get_alt_location(),
                                    atom.get_residue_type(), atom.get_chain(), atom.get_residue_number(),
                                    atom.get_code_for_insertion(), x_transformed, y_transformed, z_transformed,
                                    atom.get_occ(), atom.get_temp_fact(), atom.get_element_symbol(),
                                    atom.get_atom_charge())
            transformed_atoms.append(atom_transformed)
            atom_index += 1

        transformed_pdb.set_all_atoms(transformed_atoms)
        transformed_pdb.set_chain_id_list(pdb.get_chain_id_list())
        transformed_pdb.set_filepath(pdb.get_filepath())

        return transformed_pdb

    else:
        return []


def get_contacting_asu(pdb1, pdb2, contact_dist=8):
    pdb1_ca_coords_chain_dict = {}
    for atom in pdb1.get_all_atoms():
        if atom.chain not in pdb1_ca_coords_chain_dict:
            pdb1_ca_coords_chain_dict[atom.chain] = [atom.coords()]
        else:
            pdb1_ca_coords_chain_dict[atom.chain].append(atom.coords())

    pdb2_ca_coords_chain_dict = {}
    for atom in pdb2.get_all_atoms():
        if atom.chain not in pdb2_ca_coords_chain_dict:
            pdb2_ca_coords_chain_dict[atom.chain] = [atom.coords()]
        else:
            pdb2_ca_coords_chain_dict[atom.chain].append(atom.coords())

    max_contact_count = 0
    max_contact_chain1 = None
    max_contact_chain2 = None
    for chain1 in pdb1_ca_coords_chain_dict:
```

**utils/PDBUtils.py**

```
        for chain2 in pdb2_ca_coords_chain_dict:
            pdb1_ca_coords = pdb1_ca_coords_chain_dict[chain1]
            pdb2_ca_coords = pdb2_ca_coords_chain_dict[chain2]

            pdb1_ca_coords_kdtree = sklearn.neighbors.BallTree(np.array(pdb1_ca_coords))
            contact_count = pdb1_ca_coords_kdtree.two_point_correlation(pdb2_ca_coords, [contact_dist])[0]

            if contact_count > max_contact_count:
                max_contact_count = contact_count
                max_contact_chain1 = chain1
                max_contact_chain2 = chain2

    if max_contact_count > 0 and max_contact_chain1 is not None and max_contact_chain2 is not None:
        pdb1_asu = PDB()
        pdb1_asu.read_atom_list(pdb1.chain(max_contact_chain1))

        pdb2_asu = PDB()
        pdb2_asu.read_atom_list(pdb2.chain(max_contact_chain2))

        return pdb1_asu, pdb2_asu

    else:
        return None, None


def get_interface_ghost_surf_frags(pdb1, pdb2, pdb1_ghost_frag_list, pdb2_surf_frag_list, rot_mat1, rot_mat2,
internal_tx_vec1, internal_tx_vec2, set_mat1, set_mat2, ext_tx_vec1, ext_tx_vec2, cb_distance=9.0):

    interface_ghost_frag_list = []
    interface_ghost_frag_transformed_list = []

    interface_ghost_frag_pdb_coords_list = []
    interface_ghost_frag_pdb_coords_list_transformed = []
    interface_ghost_frag_guide_coords_list_transformed = []

    interface_surf_frag_list = []
    interface_surf_frag_transformed_list = []

    interface_surf_frag_pdb_coords_list = []
    interface_surf_frag_pdb_coords_list_transformed = []
    interface_surf_frag_guide_coords_list = []
    interface_surf_frag_guide_coords_list_transformed = []

    pdb1_cb_coords, pdb1_cb_indices = pdb1.get_CB_coords(ReturnWithCBIndices=True, InclGlyCA=True)
    pdb2_cb_coords, pdb2_cb_indices = pdb2.get_CB_coords(ReturnWithCBIndices=True, InclGlyCA=True)

    pdb1_cb_kdtree = sklearn.neighbors.BallTree(np.array(pdb1_cb_coords))

    # Query PDB1 CB Tree for all PDB2 CB Atoms within "cb_distance" in A of a PDB1 CB Atom
    query = pdb1_cb_kdtree.query_radius(pdb2_cb_coords, cb_distance)

    # Get ResidueNumber, ChainID for all Interacting PDB1 CB, PDB2 CB Pairs
    interacting_pairs = []
    for pdb2_query_index in range(len(query)):
        if query[pdb2_query_index].tolist() != list():
            pdb2_cb_res_num = pdb2.all_atoms[pdb2_cb_indices[pdb2_query_index]].residue_number
            pdb2_cb_chain_id = pdb2.all_atoms[pdb2_cb_indices[pdb2_query_index]].chain
            for pdb1_query_index in query[pdb2_query_index]:
                pdb1_cb_res_num = pdb1.all_atoms[pdb1_cb_indices[pdb1_query_index]].residue_number
                pdb1_cb_chain_id = pdb1.all_atoms[pdb1_cb_indices[pdb1_query_index]].chain
                interacting_pairs.append(((pdb1_cb_res_num, pdb1_cb_chain_id), (pdb2_cb_res_num, pdb2_cb_chain_id)))

    pdb1_central_resnum_chainid_unique_list = []
    pdb2_central_resnum_chainid_unique_list = []
    for pair in interacting_pairs:

        pdb1_central_res_num = pair[0][0]
        pdb1_central_chain_id = pair[0][1]
        pdb2_central_res_num = pair[1][0]
        pdb2_central_chain_id = pair[1][1]

        pdb1_res_num_list = [pdb1_central_res_num - 2, pdb1_central_res_num - 1, pdb1_central_res_num,
                             pdb1_central_res_num + 1, pdb1_central_res_num + 2]
        pdb2_res_num_list = [pdb2_central_res_num - 2, pdb2_central_res_num - 1, pdb2_central_res_num,
                             pdb2_central_res_num + 1, pdb2_central_res_num + 2]

        frag1_ca_count = 0
        for atom in pdb1.all_atoms:
            if atom.chain == pdb1_central_chain_id:
                if atom.residue_number in pdb1_res_num_list:
                    if atom.is_CA():
```

utils/PDBUtils.py

```
                        frag1_ca_count += 1

        frag2_ca_count = 0
        for atom in pdb2.all_atoms:
            if atom.chain == pdb2_central_chain_id:
                if atom.residue_number in pdb2_res_num_list:
                    if atom.is_CA():
                        frag2_ca_count += 1

        if frag1_ca_count == 5 and frag2_ca_count == 5:
            if (pdb1_central_chain_id, pdb1_central_res_num) not in pdb1_central_resnum_chainid_unique_list:
                pdb1_central_resnum_chainid_unique_list.append((pdb1_central_chain_id, pdb1_central_res_num))

            if (pdb2_central_chain_id, pdb2_central_res_num) not in pdb2_central_resnum_chainid_unique_list:
                pdb2_central_resnum_chainid_unique_list.append((pdb2_central_chain_id, pdb2_central_res_num))

    for ghost_frag in pdb1_ghost_frag_list:
        if ghost_frag.get_aligned_surf_frag_central_res_tup() in pdb1_central_resnum_chainid_unique_list:
            interface_ghost_frag_list.append(ghost_frag)
            interface_ghost_frag_pdb_coords_list.append(ghost_frag.get_pdb_coords())

    for surf_frag in pdb2_surf_frag_list:
        if surf_frag.get_central_res_tup() in pdb2_central_resnum_chainid_unique_list:
            interface_surf_frag_list.append(surf_frag)
            interface_surf_frag_pdb_coords_list.append(surf_frag.get_pdb_coords())
            interface_surf_frag_guide_coords_list.append(surf_frag.get_guide_coords())

    # Rotate, Translate and Set Ghost Fragment Guide Coordinates

    interface_ghost_frag_pdb_coords_list_transformed = rot_txint_set_txext_frag_coord_sets(
interface_ghost_frag_pdb_coords_list, rot_mat=rot_mat1, internal_tx_vec=internal_tx_vec1, set_mat=set_mat1,
ext_tx_vec=ext_tx_vec1)

    for int_ghost_frag_index in range(len(interface_ghost_frag_list)):
        int_ghost_frag = interface_ghost_frag_list[int_ghost_frag_index]
        int_ghost_frag_pdb = int_ghost_frag.get_pdb()
        int_ghost_frag_pdb_atoms = int_ghost_frag_pdb.get_all_atoms()

        int_ghost_frag_transformed_pdb_coords = interface_ghost_frag_pdb_coords_list_transformed[int_ghost_frag_index
]
        int_ghost_frag_pdb_transformed = PDB()
        int_ghost_frag_pdb_transformed_atoms = []
        for atom_index in range(len(int_ghost_frag_pdb_atoms)):
            atom = int_ghost_frag_pdb_atoms[atom_index]
            x_transformed = int_ghost_frag_transformed_pdb_coords[atom_index][0]
            y_transformed = int_ghost_frag_transformed_pdb_coords[atom_index][1]
            z_transformed = int_ghost_frag_transformed_pdb_coords[atom_index][2]
            atom_transformed = Atom(atom.get_number(), atom.get_type(), atom.get_alt_location(),
                                    atom.get_residue_type(), atom.get_chain(), atom.get_residue_number(),
                                    atom.get_code_for_insertion(), x_transformed, y_transformed, z_transformed,
                                    atom.get_occ(), atom.get_temp_fact(), atom.get_element_symbol(),
                                    atom.get_atom_charge())
            int_ghost_frag_pdb_transformed_atoms.append(atom_transformed)
        int_ghost_frag_pdb_transformed.set_all_atoms(int_ghost_frag_pdb_transformed_atoms)

        int_ghost_frag_transformed = GhostFragment(int_ghost_frag_pdb_transformed, int_ghost_frag.get_i_frag_type(),
                                                   int_ghost_frag.get_j_frag_type(), int_ghost_frag.get_k_frag_type()
,
                                                   int_ghost_frag.get_central_res_tup(),
                                                   int_ghost_frag.get_aligned_surf_frag_central_res_tup(),
                                                   guide_atoms=int_ghost_frag_pdb_transformed_atoms[-3:],
                                                   guide_coords=int_ghost_frag_transformed_pdb_coords[-3:],
                                                   pdb_coords=int_ghost_frag_transformed_pdb_coords)

        interface_ghost_frag_transformed_list.append(int_ghost_frag_transformed)
        interface_ghost_frag_guide_coords_list_transformed.append(int_ghost_frag_transformed.get_guide_coords())

    # Rotate, Translate and Set Surface Fragment Guide Coordinates
    interface_surf_frag_pdb_coords_list_transformed = rot_txint_set_txext_frag_coord_sets(
interface_surf_frag_pdb_coords_list, rot_mat=rot_mat2, internal_tx_vec=internal_tx_vec2, set_mat=set_mat2, ext_tx_vec
=ext_tx_vec2)
    interface_surf_frag_guide_coords_list_transformed = rot_txint_set_txext_frag_coord_sets(
interface_surf_frag_guide_coords_list, rot_mat=rot_mat2, internal_tx_vec=internal_tx_vec2, set_mat=set_mat2,
ext_tx_vec=ext_tx_vec2)

    for int_surf_frag_index in range(len(interface_surf_frag_list)):
        int_surf_frag = interface_surf_frag_list[int_surf_frag_index]

        int_surf_frag_pdb = int_surf_frag.get_pdb()
        int_surf_frag_pdb_transformed = PDB()
        int_surf_frag_transformed_pdb_coords = interface_surf_frag_pdb_coords_list_transformed[int_surf_frag_index]
```

**utils/PDBUtils.py**

```
        int_surf_frag_transformed_pdb_atoms = []
        int_surf_frag_pdb_atoms = int_surf_frag_pdb.get_all_atoms()
        for atom_index in range(len(int_surf_frag_pdb_atoms)):
            atom = int_surf_frag_pdb_atoms[atom_index]
            x_transformed = int_surf_frag_transformed_pdb_coords[atom_index][0]
            y_transformed = int_surf_frag_transformed_pdb_coords[atom_index][1]
            z_transformed = int_surf_frag_transformed_pdb_coords[atom_index][2]
            atom_transformed = Atom(atom.get_number(), atom.get_type(), atom.get_alt_location(),
                                    atom.get_residue_type(), atom.get_chain(), atom.get_residue_number(),
                                    atom.get_code_for_insertion(), x_transformed, y_transformed, z_transformed,
                                    atom.get_occ(), atom.get_temp_fact(), atom.get_element_symbol(),
                                    atom.get_atom_charge())
            int_surf_frag_transformed_pdb_atoms.append(atom_transformed)
        int_surf_frag_pdb_transformed.set_all_atoms(int_surf_frag_transformed_pdb_atoms)

        int_surf_frag_guide_coords_transformed = interface_surf_frag_guide_coords_list_transformed[
int_surf_frag_index]

        int_surf_frag_transformed = MonoFragment(int_surf_frag_pdb_transformed, type=int_surf_frag.get_type(),
                                                 guide_coords=int_surf_frag_guide_coords_transformed,
                                                 central_res_num=int_surf_frag.get_central_res_num(),
                                                 central_res_chain_id=int_surf_frag.get_central_res_chain_id(),
                                                 pdb_coords=int_surf_frag_transformed_pdb_coords)

        interface_surf_frag_transformed_list.append(int_surf_frag_transformed)

    return interface_ghost_frag_transformed_list, interface_surf_frag_transformed_list,
interface_ghost_frag_guide_coords_list_transformed, interface_surf_frag_guide_coords_list_transformed, len(
pdb1_central_resnum_chainid_unique_list), len(pdb2_central_resnum_chainid_unique_list)
```

**utils/PDBUtils.py**

BioPDBUtils.py

```
import Bio.PDB.Superimposer
from Bio.PDB.Atom import Atom as BioPDBAtom
import numpy as np
import warnings
from Bio.PDB.Atom import PDBConstructionWarning
from classes.PDB import PDB
from classes.Atom import Atom
warnings.simplefilter('ignore', PDBConstructionWarning)


def biopdb_aligned_chain(pdb_fixed, chain_id_fixed, pdb_moving, chain_id_moving):
    biopdb_atom_fixed = []
    biopdb_atom_moving = []

    for atom in pdb_fixed.get_CA_atoms():
        if atom.chain == chain_id_fixed:
            biopdb_atom_fixed.append(
                BioPDBAtom(atom.type, (atom.x, atom.y, atom.z), atom.temp_fact, atom.occ, atom.alt_location,
                           " %s " % atom.type, atom.number, element=atom.element_symbol))

    pdb_moving_coords = []
    for atom in pdb_moving.get_all_atoms():
        pdb_moving_coords.append([atom.get_x(), atom.get_y(), atom.get_z()])
        if atom.is_CA():
            if atom.chain == chain_id_moving:
                biopdb_atom_moving.append(
                    BioPDBAtom(atom.type, (atom.x, atom.y, atom.z), atom.temp_fact, atom.occ, atom.alt_location,
                               " %s " % atom.type, atom.number, element=atom.element_symbol))

    sup = Bio.PDB.Superimposer()
    sup.set_atoms(biopdb_atom_fixed, biopdb_atom_moving)
    # no need to transpose rotation matrix as Bio.PDB.Superimposer() generates correct matrix to rotate using np.
matmul
    rot, tr = sup.rotran[0], sup.rotran[1]

    pdb_moving_coords_rot = np.matmul(pdb_moving_coords, rot)
    pdb_moving_coords_rot_tx = pdb_moving_coords_rot + tr

    pdb_moving_copy = PDB()
    pdb_moving_copy.set_chain_id_list(pdb_moving.get_chain_id_list())
    pdb_moving_copy_atom_list = []
    atom_count = 0
    for atom in pdb_moving.get_all_atoms():
        x_transformed = pdb_moving_coords_rot_tx[atom_count][0]
        y_transformed = pdb_moving_coords_rot_tx[atom_count][1]
        z_transformed = pdb_moving_coords_rot_tx[atom_count][2]
        atom_transformed = Atom(atom.get_number(), atom.get_type(), atom.get_alt_location(),
                                atom.get_residue_type(), atom.get_chain(),
                                atom.get_residue_number(),
                                atom.get_code_for_insertion(), x_transformed, y_transformed,
                                z_transformed,
                                atom.get_occ(), atom.get_temp_fact(), atom.get_element_symbol(),
                                atom.get_atom_charge())
        pdb_moving_copy_atom_list.append(atom_transformed)
        atom_count += 1

    pdb_moving_copy.set_all_atoms(pdb_moving_copy_atom_list)

    return pdb_moving_copy


def biopdb_superimposer(atoms_fixed, atoms_moving):

    biopdb_atom_fixed = []
    for atom in atoms_fixed:
        biopdb_atom_fixed.append(
            BioPDBAtom(atom.type, (atom.x, atom.y, atom.z), atom.temp_fact, atom.occ, atom.alt_location,
                       " %s " % atom.type, atom.number, element=atom.element_symbol))

    biopdb_atom_moving = []
    for atom in atoms_moving:
        biopdb_atom_moving.append(
            BioPDBAtom(atom.type, (atom.x, atom.y, atom.z), atom.temp_fact, atom.occ, atom.alt_location,
                       " %s " % atom.type, atom.number, element=atom.element_symbol))

    sup = Bio.PDB.Superimposer()
    sup.set_atoms(biopdb_atom_fixed, biopdb_atom_moving)

    rmsd = sup.rms
    rot = np.transpose(sup.rotran[0]).tolist()
    tx = sup.rotran[1].tolist()
```

**utils/BioPDBUtils.py**

```
    return rmsd, rot, tx
```

**utils/BioPDBUtils.py**

GeneralUtils.py

```
import numpy as np


# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"


def euclidean_squared_3d(coordinates_1, coordinates_2):
    if len(coordinates_1) != 3 or len(coordinates_2) != 3:
        raise ValueError("len(coordinate list) != 3")

    elif type(coordinates_1) is not list or type(coordinates_2) is not list:
        raise TypeError("input parameters are not of type list")

    else:
        x1, y1, z1 = coordinates_1[0], coordinates_1[1], coordinates_1[2]
        x2, y2, z2 = coordinates_2[0], coordinates_2[1], coordinates_2[2]
        return (x1 - x2) ** 2 + (y1 - y2) ** 2 + (z1 - z2) ** 2


def center_of_mass_3d(coordinates):
    n = len(coordinates)
    if n != 0:
        cm = [0. for j in range(3)]
        for i in range(n):
            for j in range(3):
                cm[j] = cm[j] + coordinates[i][j]
        for j in range(3):
            cm[j] = cm[j] / n
        return cm
    else:
        print "ERROR CALCULATING CENTER OF MASS"
        return None


def rot_txint_set_txext_frag_coord_sets(coord_sets, rot_mat=None, internal_tx_vec=None, set_mat=None, ext_tx_vec=None
):

    if coord_sets != list():
        # Get the length of each coordinate set
        coord_set_lens = []
        for coord_set in coord_sets:
            coord_set_lens.append(len(coord_set))

        # Stack coordinate set arrays in sequence vertically (row wise)
        coord_sets_vstacked = np.vstack(coord_sets)

        # Rotate stacked coordinates if rotation matrix is provided
        if rot_mat is not None:
            rot_mat_T = np.transpose(rot_mat)
            coord_sets_vstacked = np.matmul(coord_sets_vstacked, rot_mat_T)

        # Translate stacked coordinates if internal translation vector is provided
        if internal_tx_vec is not None:
            coord_sets_vstacked = coord_sets_vstacked + internal_tx_vec

        # Set stacked coordinates if setting matrix is provided
        if set_mat is not None:
            set_mat_T = np.transpose(set_mat)
            coord_sets_vstacked = np.matmul(coord_sets_vstacked, set_mat_T)

        # Translate stacked coordinates if external translation vector is provided
        if ext_tx_vec is not None:
            coord_sets_vstacked = coord_sets_vstacked + ext_tx_vec

        # Slice stacked coordinates back into coordinate sets
        transformed_coord_sets = []
        slice_index_1 = 0
        for coord_set_len in coord_set_lens:
            slice_index_2 = slice_index_1 + coord_set_len

            transformed_coord_sets.append(coord_sets_vstacked[slice_index_1:slice_index_2].tolist())

            slice_index_1 += coord_set_len

        return transformed_coord_sets

    else:
        return []
```

**utils/GeneralUtils.py**

SymmUtils.py

```
# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"


def parse_uc_str_to_tuples(uc_string):
    return_list = []

    def s_to_l(string):
        s1 = string.replace('(', '')
        s2 = s1.replace(')', '')
        l1 = s2.split(',')
        l2 = [x.replace(' ', '') for x in l1]
        return l2

    if '),' in uc_string:
        l = uc_string.split('),')
    else:
        l = [uc_string]
    for s in l:
        return_list.append(s_to_l(s))
    return return_list


def get_uc_var_vec(string_vec, var):
    return_vec = [0.0, 0.0, 0.0]
    for i in range(len(string_vec)):
        if var in string_vec[i] and '*' in string_vec[i]:
            return_vec[i] = (float(string_vec[i].split('*')[0]))
        elif var == string_vec[i]:
            return_vec.append(1.0)
    return return_vec


def get_uc_dimensions(uc_string, e=1, f=0, g=0):
    uc_string_vec = parse_uc_str_to_tuples(uc_string)

    lengths = [0.0, 0.0, 0.0]
    string_vec_lens = uc_string_vec[0]
    e_vec = get_uc_var_vec(string_vec_lens, 'e')
    f_vec = get_uc_var_vec(string_vec_lens, 'f')
    g_vec = get_uc_var_vec(string_vec_lens, 'g')
    e1 = [e_vec_val * e for e_vec_val in e_vec]
    f1 = [f_vec_val * f for f_vec_val in f_vec]
    g1 = [g_vec_val * g for g_vec_val in g_vec]
    for i in range(len(string_vec_lens)):
        lengths[i] = abs((e1[i] + f1[i] + g1[i]))
    if len(string_vec_lens) == 2:
        lengths[2] = 1.0

    string_vec_angles = uc_string_vec[1]
    if len(string_vec_angles) == 1:
        angles = [90.0, 90.0, float(string_vec_angles[0])]
    else:
        angles = [0.0, 0.0, 0.0]
        for i in range(len(string_vec_angles)):
            angles[i] = float(string_vec_angles[i])

    uc_dimensions = lengths + angles

    return uc_dimensions
```

**utils/SymmUtils.py**

SymQueryUtils.py

```python
from classes.SymEntry import sym_comb_dict


# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"


query_out_format_str = "{:>5s} {:>6s} {:^9s} {:^9s} {:^20s} {:>6s} {:^9s} {:^9s} {:^20s} {:>6s} {:>3s} {:>4s} {:>4s}"


def print_query_header():
    header_format_str = "{:5s} {:6s} {:^9s} {:^9s} {:^20s} {:6s} {:^9s} {:^9s} {:^20s} {:6s} {:3s} {:4s} {:4s}"
    print header_format_str.format("ENTRY",
                                   "PtGrp1",
                                   "IntRtDOF1",
                                   "IntTxDOF1",
                                   "ReferenceFrameDOF1",
                                   "PtGrp2",
                                   "IntRtDOF2",
                                   "IntTxDOF2",
                                   "ReferenceFrameDOF2",
                                   "RESULT",
                                   "DIM",
                                   "#DOF",
                                   "RING")


def query_combination(combination_list):
    if type(combination_list) == list and len(combination_list) == 2:
        matching_entries = []
        for entry_number in sym_comb_dict:
            group1 = sym_comb_dict[entry_number][1]
            group2 = sym_comb_dict[entry_number][6]
            if combination_list == [group1, group2] or combination_list == [group2, group1]:
                int_rot1 = "none"
                int_tx1 = "none"
                int_rot2 = "none"
                int_tx2 = "none"
                int_dof_group1 = sym_comb_dict[entry_number][3]
                int_dof_group2 = sym_comb_dict[entry_number][8]
                for int_dof in int_dof_group1:
                    if int_dof.startswith('r'):
                        int_rot1 = int_dof[2:]
                    if int_dof.startswith('t'):
                        int_tx1 = int_dof[2:]
                for int_dof in int_dof_group2:
                    if int_dof.startswith('r'):
                        int_rot2 = int_dof[2:]
                    if int_dof.startswith('t'):
                        int_tx2 = int_dof[2:]
                ref_frame_tx_dof_group1 = sym_comb_dict[entry_number][5]
                ref_frame_tx_dof_group2 = sym_comb_dict[entry_number][10]
                if ref_frame_tx_dof_group1 == '<0,0,0>':
                    ref_frame_tx_dof_group1 = 'none'
                if ref_frame_tx_dof_group2 == '<0,0,0>':
                    ref_frame_tx_dof_group2 = 'none'
                result = sym_comb_dict[entry_number][12]
                dim = sym_comb_dict[entry_number][13]
                tot_num_dof = sym_comb_dict[entry_number][15]
                ring_size = sym_comb_dict[entry_number][16]
                matching_entries.append(query_out_format_str.format(str(entry_number),
                                                                    group1,
                                                                    int_rot1,
                                                                    int_tx1,
                                                                    ref_frame_tx_dof_group1,
                                                                    group2,
                                                                    int_rot2,
                                                                    int_tx2,
                                                                    ref_frame_tx_dof_group2,
                                                                    result,
                                                                    str(dim),
                                                                    str(tot_num_dof),
                                                                    str(ring_size)))
        if matching_entries == list():
            print '\033[1m' + "NO MATCHING ENTRY FOUND" + '\033[0m'
            print ''
        else:
            print '\033[1m' + "POSSIBLE COMBINATION(S) FOR: %s & %s" % (combination_list[0], combination_list[1]) + '
\033[0m'
```

**utils/SymQueryUtils.py**

```
            print_query_header()
            for match in matching_entries:
                print match
    else:
        print "INVALID ENTRY"


def query_result(desired_result):
    if type(desired_result) == str:
        matching_entries = []
        for entry_number in sym_comb_dict:
            result = sym_comb_dict[entry_number][12]
            if desired_result == result:
                group1 = sym_comb_dict[entry_number][1]
                group2 = sym_comb_dict[entry_number][6]
                int_rot1 = "none"
                int_tx1 = "none"
                int_rot2 = "none"
                int_tx2 = "none"
                int_dof_group1 = sym_comb_dict[entry_number][3]
                int_dof_group2 = sym_comb_dict[entry_number][8]
                for int_dof in int_dof_group1:
                    if int_dof.startswith('r'):
                        int_rot1 = int_dof[2:]
                    if int_dof.startswith('t'):
                        int_tx1 = int_dof[2:]
                for int_dof in int_dof_group2:
                    if int_dof.startswith('r'):
                        int_rot2 = int_dof[2:]
                    if int_dof.startswith('t'):
                        int_tx2 = int_dof[2:]
                ref_frame_tx_dof_group1 = sym_comb_dict[entry_number][5]
                ref_frame_tx_dof_group2 = sym_comb_dict[entry_number][10]
                if ref_frame_tx_dof_group1 == '<0,0,0>':
                    ref_frame_tx_dof_group1 = 'none'
                if ref_frame_tx_dof_group2 == '<0,0,0>':
                    ref_frame_tx_dof_group2 = 'none'
                dim = sym_comb_dict[entry_number][13]
                tot_num_dof = sym_comb_dict[entry_number][15]
                ring_size = sym_comb_dict[entry_number][16]
                matching_entries.append(query_out_format_str.format(str(entry_number),
                                                                    group1,
                                                                    int_rot1,
                                                                    int_tx1,
                                                                    ref_frame_tx_dof_group1,
                                                                    group2,
                                                                    int_rot2,
                                                                    int_tx2,
                                                                    ref_frame_tx_dof_group2,
                                                                    result,
                                                                    str(dim),
                                                                    str(tot_num_dof),
                                                                    str(ring_size)))
        if matching_entries == list():
            print '\033[1m' + "NO MATCHING ENTRY FOUND" + '\033[0m'
            print ''
        else:
            print '\033[1m' + "POSSIBLE COMBINATION(S) FOR: %s" % desired_result + '\033[0m'
            print_query_header()
            for match in matching_entries:
                print match
    else:
        print "INVALID ENTRY"


def query_counterpart(query_group):
    if type(query_group) == str:
        matching_entries = []
        for entry_number in sym_comb_dict:
            group1 = sym_comb_dict[entry_number][1]
            group2 = sym_comb_dict[entry_number][6]
            if query_group in [group1, group2]:
                int_rot1 = "none"
                int_tx1 = "none"
                int_rot2 = "none"
                int_tx2 = "none"
                int_dof_group1 = sym_comb_dict[entry_number][3]
                int_dof_group2 = sym_comb_dict[entry_number][8]
                for int_dof in int_dof_group1:
                    if int_dof.startswith('r'):
                        int_rot1 = int_dof[2:]
```

**utils/SymQueryUtils.py**

```
                    if int_dof.startswith('t'):
                        int_tx1 = int_dof[2:]
                for int_dof in int_dof_group2:
                    if int_dof.startswith('r'):
                        int_rot2 = int_dof[2:]
                    if int_dof.startswith('t'):
                        int_tx2 = int_dof[2:]
                ref_frame_tx_dof_group1 = sym_comb_dict[entry_number][5]
                ref_frame_tx_dof_group2 = sym_comb_dict[entry_number][10]
                if ref_frame_tx_dof_group1 == '<0,0,0>':
                    ref_frame_tx_dof_group1 = 'none'
                if ref_frame_tx_dof_group2 == '<0,0,0>':
                    ref_frame_tx_dof_group2 = 'none'
                result = sym_comb_dict[entry_number][12]
                dim = sym_comb_dict[entry_number][13]
                tot_num_dof = sym_comb_dict[entry_number][15]
                ring_size = sym_comb_dict[entry_number][16]
                matching_entries.append(query_out_format_str.format(str(entry_number),
                                                                    group1,
                                                                    int_rot1,
                                                                    int_tx1,
                                                                    ref_frame_tx_dof_group1,
                                                                    group2,
                                                                    int_rot2,
                                                                    int_tx2,
                                                                    ref_frame_tx_dof_group2,
                                                                    result,
                                                                    str(dim),
                                                                    str(tot_num_dof),
                                                                    str(ring_size)))
        if matching_entries == list():
            print '\033[1m' + "NO MATCHING ENTRY FOUND" + '\033[0m'
            print ''
        else:
            print '\033[1m' + "POSSIBLE COMBINATION(S) FOR: %s" % query_group + '\033[0m'
            print_query_header()
            for match in matching_entries:
                print match
    else:
        print "INVALID ENTRY"


def all_entries():
    all_entries_list = []
    for entry_number in sym_comb_dict:
        group1 = sym_comb_dict[entry_number][1]
        group2 = sym_comb_dict[entry_number][6]
        int_rot1 = "none"
        int_tx1 = "none"
        int_rot2 = "none"
        int_tx2 = "none"
        int_dof_group1 = sym_comb_dict[entry_number][3]
        int_dof_group2 = sym_comb_dict[entry_number][8]
        for int_dof in int_dof_group1:
            if int_dof.startswith('r'):
                int_rot1 = int_dof[2:]
            if int_dof.startswith('t'):
                int_tx1 = int_dof[2:]
        for int_dof in int_dof_group2:
            if int_dof.startswith('r'):
                int_rot2 = int_dof[2:]
            if int_dof.startswith('t'):
                int_tx2 = int_dof[2:]
        ref_frame_tx_dof_group1 = sym_comb_dict[entry_number][5]
        ref_frame_tx_dof_group2 = sym_comb_dict[entry_number][10]
        if ref_frame_tx_dof_group1 == '<0,0,0>':
            ref_frame_tx_dof_group1 = 'none'
        if ref_frame_tx_dof_group2 == '<0,0,0>':
            ref_frame_tx_dof_group2 = 'none'
        result = sym_comb_dict[entry_number][12]
        dim = sym_comb_dict[entry_number][13]
        tot_num_dof = sym_comb_dict[entry_number][15]
        ring_size = sym_comb_dict[entry_number][16]
        all_entries_list.append(query_out_format_str.format(str(entry_number),
                                                            group1,
                                                            int_rot1,
                                                            int_tx1,
                                                            ref_frame_tx_dof_group1,
                                                            group2,
                                                            int_rot2,
                                                            int_tx2,
```

**utils/SymQueryUtils.py**

```
                                                              ref_frame_tx_dof_group2,
                                                              result,
                                                              str(dim),
                                                              str(tot_num_dof),
                                                              str(ring_size)))
    print '\033[1m' + "ALL ENTRIES" + '\033[0m'
    print_query_header()
    for entry in all_entries_list:
        print entry


def dimension(dim):
    if dim in [0, 2, 3]:
        matching_entries = []
        for entry_number in sym_comb_dict:
            if sym_comb_dict[entry_number][13] == dim:
                group1 = sym_comb_dict[entry_number][1]
                group2 = sym_comb_dict[entry_number][6]
                int_rot1 = "none"
                int_tx1 = "none"
                int_rot2 = "none"
                int_tx2 = "none"
                int_dof_group1 = sym_comb_dict[entry_number][3]
                int_dof_group2 = sym_comb_dict[entry_number][8]
                for int_dof in int_dof_group1:
                    if int_dof.startswith('r'):
                        int_rot1 = int_dof[2:]
                    if int_dof.startswith('t'):
                        int_tx1 = int_dof[2:]
                for int_dof in int_dof_group2:
                    if int_dof.startswith('r'):
                        int_rot2 = int_dof[2:]
                    if int_dof.startswith('t'):
                        int_tx2 = int_dof[2:]
                ref_frame_tx_dof_group1 = sym_comb_dict[entry_number][5]
                ref_frame_tx_dof_group2 = sym_comb_dict[entry_number][10]
                if ref_frame_tx_dof_group1 == '<0,0,0>':
                    ref_frame_tx_dof_group1 = 'none'
                if ref_frame_tx_dof_group2 == '<0,0,0>':
                    ref_frame_tx_dof_group2 = 'none'
                result = sym_comb_dict[entry_number][12]
                dim = sym_comb_dict[entry_number][13]
                tot_num_dof = sym_comb_dict[entry_number][15]
                ring_size = sym_comb_dict[entry_number][16]
                matching_entries.append(query_out_format_str.format(str(entry_number),
                                                                    group1,
                                                                    int_rot1,
                                                                    int_tx1,
                                                                    ref_frame_tx_dof_group1,
                                                                    group2,
                                                                    int_rot2,
                                                                    int_tx2,
                                                                    ref_frame_tx_dof_group2,
                                                                    result,
                                                                    str(dim),
                                                                    str(tot_num_dof),
                                                                    str(ring_size)))

        print '\033[1m' + "ALL ENTRIES FOUND WITH DIMENSION " + str(dim) + ": " + '\033[0m'
        print_query_header()
        for entry in matching_entries:
            print entry
    else:
        print "DIMENSION NOT SUPPORTED, VALID DIMENSIONS ARE: 0, 2 or 3 "
```

**utils/SymQueryUtils.py**

CmdLineArgParseUtils.py

```
from utils import SymQueryUtils
from utils import PostProcessUtils
import os
import sys
from classes.SymEntry import SymEntry


# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"


def query_mode(arg_list):
    valid_query_flags = ["-all_entries", "-combination", "-result", "-counterpart", "-dimension"]
    if len(arg_list) >= 3 and arg_list[1] == "-query" and arg_list[2] in valid_query_flags:
        print '\033[32m' + '\033[1m' + "NANOHEDRA\n" + '\033[0m'
        print '\033[32m' + '\033[1m' + "Copyright 2020 Joshua Laniado and Todd O. Yeates\n\n" + '\033[0m'
        print '\033[1m' + '\033[95m' + "QUERY MODE" + '\033[95m' + '\033[0m' + '\n'
        if arg_list[2] == "-all_entries":
            if len(arg_list) == 3:
                SymQueryUtils.all_entries()
            else:
                sys.exit('\033[91m' + '\033[1m' + "ERROR: INVALID QUERY" + '\033[0m')

        elif arg_list[2] == "-combination":
            if len(arg_list) == 5:
                query = [arg_list[3], arg_list[4]]
                SymQueryUtils.query_combination(query)
            else:
                sys.exit('\033[91m' + '\033[1m' + "ERROR: INVALID COMBINATION QUERY" + '\033[0m')

        elif arg_list[2] == "-result":
            if len(arg_list) == 4:
                query = arg_list[3]
                SymQueryUtils.query_result(query)
            else:
                sys.exit('\033[91m' + '\033[1m' + "ERROR: INVALID RESULT QUERY" + '\033[0m')

        elif arg_list[2] == "-counterpart":
            if len(arg_list) == 4:
                query = arg_list[3]
                SymQueryUtils.query_counterpart(query)
            else:
                sys.exit('\033[91m' + '\033[1m' + "ERROR: INVALID COUNTERPART QUERY" + '\033[0m')

        elif arg_list[2] == "-dimension":
            if len(arg_list) == 4 and arg_list[3].isdigit():
                query = int(arg_list[3])
                SymQueryUtils.dimension(query)
            else:
                sys.exit('\033[91m' + '\033[1m' + "ERROR: INVALID QUERY" + '\033[0m')

    else:
        sys.exit('\033[91m' + '\033[1m' + "ERROR: INVALID QUERY, CHOOSE ONE OF THE FOLLOWING QUERY FLAGS: -
all_entries, -combination, -result, -counterpart, -dimension" + '\033[0m')


def get_docking_parameters(arg_list):

    if "-outdir" in arg_list:
        outdir_index = arg_list.index('-outdir') + 1
        if outdir_index < len(arg_list):
            outdir = arg_list[outdir_index]
        else:
            log_filepath = os.getcwd() + "/Nanohedra_log.txt"
            logfile = open(log_filepath, "a+")
            logfile.write("ERROR: OUTPUT DIRECTORY NOT SPECIFIED" + '\n')
            logfile.close()
            sys.exit()
    else:
        log_filepath = os.getcwd() + "/Nanohedra_log.txt"
        logfile = open(log_filepath, "a+")
        logfile.write("ERROR: OUTPUT DIRECTORY NOT SPECIFIED" + '\n')
        logfile.close()
        sys.exit()

    master_log_filepath = outdir + "/nanohedra_master_logfile.txt"
    if not os.path.exists(outdir):
        os.makedirs(outdir)
    master_logfile = open(master_log_filepath, "w")
```

**utils/CmdLineArgParseUtils.py**

```
    valid_flags = ["-dock", "-entry", "-oligomer1", "-oligomer2", "-rot_step1", "-rot_step2", "-outdir",
                   "-output_uc", "-output_surrounding_uc", "-min_matched", "-output_exp_assembly", "-init_match_type"
]

    # CHECK INPUT FLAGS
    for sys_input in arg_list:
        if sys_input.startswith('-') and sys_input not in valid_flags:
            master_logfile.write("ERROR: " + sys_input + " IS AN INVALID FLAG" + "\n")
            master_logfile.write("VALID FLAGS FOR DOCKING ARE:" + "\n")
            for flag in valid_flags:
                master_logfile.write(flag + "\n")
            master_logfile.close()
            sys.exit()

    # DOCK MODE
    master_logfile.write("NANOHEDRA" + "\n")
    master_logfile.write("DOCKING MODE" + '\n\n')

    # SymEntry PARAMETER
    if "-entry" in arg_list:
        entry_index = arg_list.index('-entry') + 1
        if entry_index < len(arg_list):
            if arg_list[entry_index].isdigit() and (int(arg_list[entry_index]) in range(1, 125)):
                entry = int(arg_list[arg_list.index('-entry') + 1])
            else:
                master_logfile.write("ERROR: INVALID SYMMETRY ENTRY. SUPPORTED VALUES ARE: 1 to 124\n")
                master_logfile.close()
                sys.exit()
        else:
            master_logfile.write("ERROR: SYMMETRY ENTRY NOT SPECIFIED\n")
            master_logfile.close()
            sys.exit()
    else:
        master_logfile.write("ERROR: SYMMETRY ENTRY NOT SPECIFIED\n")
        master_logfile.close()
        sys.exit()

    # GENERAL INPUT PARAMETERS

    # determine whether or not both components needed to construct the SCM have the same point group symmetry
    # if so, only one input PDB directory should be provided, otherwise two are required
    sym_entry = SymEntry(entry)
    oligomer_symmetry_1 = sym_entry.get_group1_sym()
    oligomer_symmetry_2 = sym_entry.get_group2_sym()

    if oligomer_symmetry_1 == oligomer_symmetry_2:
        if ("-oligomer1" in arg_list) and ("-oligomer2" in arg_list):
            master_logfile.write("ERROR\nONLY ONE INPUT PDB DIRECTORY IS ACCEPTED FOR ENTRY %s\n" % str(entry))
            master_logfile.write("BOTH COMPONENTS NEEDED TO CONSTRUCT THIS SCM HAVE THE SAME POINT GROUP SYMMETRY ")
            master_logfile.write("(%s)\n" % oligomer_symmetry_1)
            master_logfile.write("THE INPUT PDB FILES FOR THIS ENTRY SHOULD BE IN A SINGLE DIRECTORY\n")
            master_logfile.write("AND ONLY '-oligomer1' SHOULD BE USED TO SPECIFY THE INPUT PDB DIRECTORY\n")
            master_logfile.close()
            sys.exit()
        elif ("-oligomer1" not in arg_list) and ("-oligomer2" in arg_list):
            master_logfile.write("ERROR\nONLY ONE INPUT PDB DIRECTORY IS ACCEPTED FOR ENTRY %s\n" % str(entry))
            master_logfile.write("BOTH COMPONENTS NEEDED TO CONSTRUCT THIS SCM HAVE THE SAME POINT GROUP SYMMETRY ")
            master_logfile.write("(%s)\n" % oligomer_symmetry_1)
            master_logfile.write("THE INPUT PDB FILES FOR THIS ENTRY SHOULD BE IN A SINGLE DIRECTORY\n")
            master_logfile.write("AND ONLY '-oligomer1' SHOULD BE USED TO SPECIFY THE INPUT PDB DIRECTORY\n")
            master_logfile.close()
            sys.exit()
        elif ("-oligomer1" in arg_list) and ("-oligomer2" not in arg_list):
            path1_index = arg_list.index('-oligomer1') + 1

            if path1_index < len(arg_list):
                path1 = arg_list[arg_list.index('-oligomer1') + 1]
                if os.path.exists(path1):
                    pdb_dir1_path = path1
                    pdb_dir2_path = path1
                else:
                    master_logfile.write("ERROR: SPECIFIED PDB DIRECTORY PATH DOES NOT EXIST\n")
                    master_logfile.close()
                    sys.exit()
            else:
                master_logfile.write("ERROR: PDB DIRECTORY PATH NOT SPECIFIED\n")
                master_logfile.close()
                sys.exit()
        else:
            master_logfile.write("ERROR: PDB DIRECTORY PATH NOT SPECIFIED\n")
```

**utils/CmdLineArgParseUtils.py**

```
            master_logfile.close()
            sys.exit()

    else:
        if ("-oligomer1" in arg_list) and ("-oligomer2" in arg_list):
            path1_index = arg_list.index('-oligomer1') + 1
            path2_index = arg_list.index('-oligomer2') + 1

            if (path1_index < len(arg_list)) and (path2_index < len(arg_list)):
                path1 = arg_list[arg_list.index('-oligomer1') + 1]
                path2 = arg_list[arg_list.index('-oligomer2') + 1]
                if os.path.exists(path1) and os.path.exists(path2):
                    pdb_dir1_path = path1
                    pdb_dir2_path = path2
                else:
                    master_logfile.write("ERROR: SPECIFIED PDB DIRECTORY PATH(S) DO(ES) NOT EXIST" + "\n")
                    master_logfile.close()
                    sys.exit()
            else:
                master_logfile.write("ERROR: PDB DIRECTORY PATH(S) NOT SPECIFIED" + "\n")
                master_logfile.close()
                sys.exit()
        else:
            master_logfile.write("ERROR: PDB DIRECTORY PATH(S) NOT SPECIFIED" + "\n")
            master_logfile.close()
            sys.exit()

    if "-init_match_type" in arg_list:
        init_match_type_index = arg_list.index('-init_match_type') + 1

        if init_match_type_index < len(arg_list):
            if arg_list[init_match_type_index] in ["1_1", "1_2", "2_1", "2_2"]:
                init_match_type = str(arg_list[init_match_type_index])
            else:
                master_logfile.write("ERROR: INITIAL FRAGMENT MATCH TYPE SPECIFIED NOT RECOGNIZED" + "\n")
                master_logfile.close()
                sys.exit()
        else:
            master_logfile.write("ERROR: INITIAL FRAGMENT MATCH TYPE NOT SPECIFIED" + "\n")
            master_logfile.close()
            sys.exit()
    else:
        init_match_type = "1_1"  # default initial fragment match type is set to helix-helix interactions ==> "1_1"

    # FragDock PARAMETERS
    if "-rot_step1" in arg_list:
        rot_step_index1 = arg_list.index('-rot_step1') + 1
        if rot_step_index1 < len(arg_list):
            if arg_list[rot_step_index1].isdigit():
                rot_step_deg1 = int(arg_list[rot_step_index1])
            else:
                master_logfile.write("ERROR: ROTATION STEP SPECIFIED IS NOT AN INTEGER" + "\n")
                master_logfile.close()
                sys.exit()
        else:
            master_logfile.write("ERROR: ROTATION STEP NOT SPECIFIED" + "\n")
            master_logfile.close()
            sys.exit()
    else:
        rot_step_deg1 = None

    if "-rot_step2" in arg_list:
        rot_step_index2 = arg_list.index('-rot_step2') + 1
        if rot_step_index2 < len(arg_list):
            if arg_list[rot_step_index2].isdigit():
                rot_step_deg2 = int(arg_list[rot_step_index2])
            else:
                master_logfile.write("ERROR: ROTATION STEP SPECIFIED IS NOT AN INTEGER" + "\n")
                master_logfile.close()
                sys.exit()
        else:
            master_logfile.write("ERROR: ROTATION STEP NOT SPECIFIED" + "\n")
            master_logfile.close()
            sys.exit()
    else:
        rot_step_deg2 = None

    if "-output_exp_assembly" in arg_list:
        output_exp_assembly = True
    else:
        output_exp_assembly = False
```

**utils/CmdLineArgParseUtils.py**

```
    if "-output_uc" in arg_list:
        output_uc = True
    else:
        output_uc = False

    if "-output_surrounding_uc" in arg_list:
        output_surrounding_uc = True
    else:
        output_surrounding_uc = False

    if "-min_matched" in arg_list:
        min_matched_index = arg_list.index('-min_matched') + 1
        if min_matched_index < len(arg_list):
            if arg_list[min_matched_index].isdigit():
                min_matched = int(arg_list[min_matched_index])
            else:
                master_logfile.write("ERROR: MINIMUM NUMBER OF REQUIRED MATCHED FRAGMENT(S) SPECIFIED IS NOT AN
INTEGER" + "\n")
                master_logfile.close()
                sys.exit()
        else:
            master_logfile.write("ERROR: MINIMUM NUMBER OF REQUIRED MATCHED FRAGMENT(S) NOT SPECIFIED" + "\n")
            master_logfile.close()
            sys.exit()
    else:
        min_matched = 3

    master_logfile.close()

    return entry, pdb_dir1_path, pdb_dir2_path, rot_step_deg1, rot_step_deg2, outdir, output_exp_assembly, output_uc,
 output_surrounding_uc, min_matched, init_match_type


def postprocess_mode(arg_list):

    valid_flags = ["-outdir", "-design_dir", "-min_score", "-min_matched", "-postprocess", "-rank"]
    for arg in arg_list:
        if arg[0] == '-' and arg not in valid_flags:
            log_filepath = os.getcwd() + "/Nanohedra_PostProcess_log.txt"
            logfile = open(log_filepath, "a+")
            logfile.write("ERROR: %s IS AN INVALID FLAG\n" %arg)
            logfile.write("VALID FLAGS ARE: -outdir, -design_dir, -min_score, -min_matched, -rank, -postprocess\n")
            logfile.close()
            sys.exit()

    if "-outdir" in arg_list:
        outdir_index = arg_list.index('-outdir') + 1
        if outdir_index < len(arg_list):
            outdir = arg_list[outdir_index]
        else:
            log_filepath = os.getcwd() + "/Nanohedra_PostProcess_log.txt"
            logfile = open(log_filepath, "a+")
            logfile.write("ERROR: OUTPUT DIRECTORY NOT SPECIFIED\n")
            logfile.close()
            sys.exit()
    else:
        log_filepath = os.getcwd() + "/Nanohedra_PostProcess_log.txt"
        logfile = open(log_filepath, "a+")
        logfile.write("ERROR: OUTPUT DIRECTORY NOT SPECIFIED\n")
        logfile.close()
        sys.exit()

    log_filepath = outdir + "/Nanohedra_PostProcess_log.txt"
    if not os.path.exists(outdir):
        os.makedirs(outdir)

    try:
        design_dir_path_index = arg_list.index("-design_dir") + 1
        try:
            design_dir_path = arg_list[design_dir_path_index]
            if not os.path.exists(design_dir_path):
                logfile = open(log_filepath, "w")
                logfile.write("ERROR: DESIGN DIRECTORY PATH SPECIFIED DOES NOT EXIST\n")
                logfile.close()
                sys.exit()
        except IndexError:
            logfile = open(log_filepath, "w")
            logfile.write("ERROR: -design_dir FLAG FOLLOWED BY DESIGN DIRECTORY PATH IS REQUIRED\n")
            logfile.close()
            sys.exit()
```

**utils/CmdLineArgParseUtils.py**

```
    except ValueError:
        logfile = open(log_filepath, "w")
        logfile.write("ERROR: -design_dir FLAG FOLLOWED BY DESIGN DIRECTORY PATH IS REQUIRED\n")
        logfile.close()
        sys.exit()

    if "-min_score" in arg_list and "-min_matched" not in arg_list and "-rank" not in arg_list:
        try:
            min_score_index = arg_list.index("-min_score") + 1
            try:
                min_score_str = arg_list[min_score_index]
                try:
                    min_score = float(min_score_str)
                    PostProcessUtils.score_filter(design_dir_path, min_score, outdir)
                except ValueError:
                    logfile = open(log_filepath, "w")
                    logfile.write("ERROR: MINIMUM SCORE SPECIFIED IS NOT A FLOAT\n")
                    logfile.close()
                    sys.exit()
            except IndexError:
                logfile = open(log_filepath, "w")
                logfile.write("ERROR: -min_score FLAG FOLLOWED BY A MINIMUM SCORE IS REQUIRED\n")
                logfile.close()
                sys.exit()
        except ValueError:
            logfile = open(log_filepath, "w")
            logfile.write("ERROR: -min_score FLAG FOLLOWED BY A MINIMUM SCORE IS REQUIRED\n")
            logfile.close()
            sys.exit()

    if "-min_matched" in arg_list and "-min_score" not in arg_list and "-rank" not in arg_list:
        try:
            min_matched_index = arg_list.index("-min_matched") + 1
            try:
                min_matched_str = arg_list[min_matched_index]
                try:
                    min_matched = int(min_matched_str)
                    PostProcessUtils.frag_match_count_filter(design_dir_path, min_matched, outdir)
                except ValueError:
                    logfile = open(log_filepath, "w")
                    logfile.write("ERROR: MINIMUM MATCHED FRAGMENT COUNT SPECIFIED IS NOT AN INTEGER\n")
                    logfile.close()
                    sys.exit()
            except IndexError:
                logfile = open(log_filepath, "w")
                logfile.write("ERROR: -min_matched FLAG FOLLOWED BY A MINIMUM MATCHED FRAGMENT COUNT IS REQUIRED\n")
                logfile.close()
                sys.exit()
        except ValueError:
            logfile = open(log_filepath, "w")
            logfile.write("ERROR: -min_matched FLAG FOLLOWED BY A MINIMUM MATCHED FRAGMENT COUNT IS REQUIRED\n")
            logfile.close()
            sys.exit()

    if "-min_matched" in arg_list and "-min_score" in arg_list and "-rank" not in arg_list:
        min_matched_index = arg_list.index("-min_matched") + 1
        min_score_index = arg_list.index("-min_score") + 1
        try:
            min_matched_str = arg_list[min_matched_index]
            min_score_str = arg_list[min_score_index]

            try:
                min_matched = int(min_matched_str)
            except ValueError:
                logfile = open(log_filepath, "w")
                logfile.write("ERROR: MINIMUM MATCHED FRAGMENT COUNT SPECIFIED IS NOT AN INTEGER\n")
                logfile.close()
                sys.exit()

            try:
                min_score = float(min_score_str)
            except ValueError:
                logfile = open(log_filepath, "w")
                logfile.write("ERROR: MINIMUM SCORE SPECIFIED IS NOT A FLOAT\n")
                logfile.close()
                sys.exit()

            PostProcessUtils.score_and_frag_match_count_filter(design_dir_path, min_score, min_matched, outdir)

        except IndexError:
            logfile = open(log_filepath, "w")
```

**utils/CmdLineArgParseUtils.py**

```
            logfile.write("ERROR: -min_matched FLAG FOLLOWED BY A MINIMUM MATCHED FRAGMENT COUNT IS REQUIRED\n")
            logfile.write("ERROR: -min_score FLAG FOLLOWED BY A MINIMUM SCORE IS REQUIRED\n")
            logfile.close()
            sys.exit()

    if "-rank" in arg_list and "-min_matched" not in arg_list and "-min_score" not in arg_list:
        try:
            rank_index = arg_list.index("-rank") + 1
            try:
                metric = arg_list[rank_index]
                try:
                    metric_str = str(metric)
                    if metric_str in ["score", "matched"]:
                        PostProcessUtils.rank(design_dir_path, metric_str, outdir)
                    else:
                        logfile = open(log_filepath, "w")
                        logfile.write("ERROR: RANKING METRIC SPECIFIED IS NOT RECOGNIZED\n")
                        logfile.close()
                        sys.exit()
                except ValueError:
                    logfile = open(log_filepath, "w")
                    logfile.write("ERROR: RANKING METRIC SPECIFIED IS NOT A STRING\n")
                    logfile.close()
                    sys.exit()
            except IndexError:
                logfile = open(log_filepath, "w")
                logfile.write("ERROR: -rank FLAG FOLLOWED BY A RANKING METRIC IS REQUIRED\n")
                logfile.close()
                sys.exit()
        except ValueError:
            logfile = open(log_filepath, "w")
            logfile.write("ERROR: -rank FLAG FOLLOWED BY A RANKING METRIC IS REQUIRED\n")
            logfile.close()
            sys.exit()

    if ("-min_matched" in arg_list or "-min_score" in arg_list) and "-rank" in arg_list:
        logfile = open(log_filepath, "w")
        logfile.write("ERROR:\n")
        logfile.write("EITHER: FILTER BY SCORE AND/OR BY MINIMUM FRAGMENT(S) MATCHED\n")
        logfile.write("OR: PERFORM RANKING BY SCORE\n")
        logfile.write("OR: PERFORM RANKING BY NUMBER OF FRAGMENT(S) MATCHED\n")
        logfile.close()
        sys.exit()

    if "-min_matched" not in arg_list and "-min_score" not in arg_list and "-rank" not in arg_list:
        logfile = open(log_filepath, "w")
        logfile.write("ERROR: POST PROCESSING FLAG REQUIRED\n")
        logfile.close()
        sys.exit()
```

**utils/CmdLineArgParseUtils.py**

PostProcessUtils.py

```python
import os
import shutil


# Copyright 2020 Joshua Laniado and Todd O. Yeates.
__author__ = "Joshua Laniado and Todd O. Yeates"
__copyright__ = "Copyright 2020, Nanohedra"
__version__ = "1.0"


def frag_match_count_filter(master_design_dirpath, min_frag_match_count, master_design_outdir_path):
    for root1, dirs1, files1 in os.walk(master_design_dirpath):
        for file1 in files1:
            if "docked_pose_info_file.txt" in file1:
                info_file_filepath = root1 + "/" + file1

                tx_filepath = root1
                rot_filepath= os.path.dirname(tx_filepath)
                degen_filepath = os.path.dirname(rot_filepath)
                design_filepath = os.path.dirname(degen_filepath)

                tx_fname = tx_filepath.split("/")[-1]
                rot_fname = rot_filepath.split("/")[-1]
                degen_fname = degen_filepath.split("/")[-1]
                design_fname = design_filepath.split("/")[-1]

                outdir = master_design_outdir_path + "/" + design_fname + "/" + degen_fname + "/" + rot_fname + "/" +
 tx_fname

                info_file = open(info_file_filepath, 'r')
                for line in info_file.readlines():
                    if "Unique Mono Fragments Matched:" in line:
                        frag_match_count = int(line[30:])
                        if frag_match_count >= min_frag_match_count:
                            shutil.copytree(tx_filepath, outdir)
                info_file.close()


def score_filter(master_design_dirpath, min_score, master_design_outdir_path):
    for root1, dirs1, files1 in os.walk(master_design_dirpath):
        for file1 in files1:
            if "docked_pose_info_file.txt" in file1:
                info_file_filepath = root1 + "/" + file1

                tx_filepath = root1
                rot_filepath = os.path.dirname(tx_filepath)
                degen_filepath = os.path.dirname(rot_filepath)
                design_filepath = os.path.dirname(degen_filepath)

                tx_fname = tx_filepath.split("/")[-1]
                rot_fname = rot_filepath.split("/")[-1]
                degen_fname = degen_filepath.split("/")[-1]
                design_fname = design_filepath.split("/")[-1]

                outdir = master_design_outdir_path + "/" + design_fname + "/" + degen_fname + "/" + rot_fname + "/" +
 tx_fname

                info_file = open(info_file_filepath, 'r')
                for line in info_file.readlines():
                    if "Nanohedra Score:" in line:
                        score = float(line[17:])
                        if score >= min_score:
                            shutil.copytree(tx_filepath, outdir)
                info_file.close()


def score_and_frag_match_count_filter(master_design_dirpath, min_score, min_frag_match_count,
master_design_outdir_path):
    for root1, dirs1, files1 in os.walk(master_design_dirpath):
        for file1 in files1:
            if "docked_pose_info_file.txt" in file1:
                info_file_filepath = root1 + "/" + file1

                tx_filepath = root1
                rot_filepath = os.path.dirname(tx_filepath)
                degen_filepath = os.path.dirname(rot_filepath)
                design_filepath = os.path.dirname(degen_filepath)

                tx_fname = tx_filepath.split("/")[-1]
                rot_fname = rot_filepath.split("/")[-1]
                degen_fname = degen_filepath.split("/")[-1]
```

**utils/PostProcessUtils.py**

```
                design_fname = design_filepath.split("/")[-1]

                outdir = master_design_outdir_path + "/" + design_fname + "/" + degen_fname + "/" + rot_fname + "/" +
 tx_fname

                score = None
                frag_match_count = None
                info_file = open(info_file_filepath, 'r')
                for line in info_file.readlines():
                    if "Nanohedra Score:" in line:
                        score = float(line[17:])
                    if "Unique Mono Fragments Matched:" in line:
                        frag_match_count = int(line[30:])
                info_file.close()

                if score is not None and frag_match_count is not None:
                    if score >= min_score and frag_match_count >= min_frag_match_count:
                        shutil.copytree(tx_filepath, outdir)


def rank(master_design_dirpath, metric, outdir):

    if metric == 'score':
        metric_str = "Nanohedra Score:"
    elif metric == 'matched':
        metric_str = "Unique Mono Fragments Matched:"
    else:
        raise ValueError('\n%s is not a recognized ranking metric. '
                         'Recognized ranking metrics are: score and matched.\n' %str(metric))

    designpath_metric_tup_list = []

    for root1, dirs1, files1 in os.walk(master_design_dirpath):
        for file1 in files1:
            if "docked_pose_info_file.txt" in file1:
                info_file_filepath = root1 + "/" + file1

                tx_filepath = root1
                rot_filepath = os.path.dirname(tx_filepath)
                degen_filepath = os.path.dirname(rot_filepath)
                design_filepath = os.path.dirname(degen_filepath)

                tx_fname = tx_filepath.split("/")[-1]
                rot_fname = rot_filepath.split("/")[-1]
                degen_fname = degen_filepath.split("/")[-1]
                design_fname = design_filepath.split("/")[-1]

                design_path = "/" + design_fname + "/" + degen_fname + "/" + rot_fname + "/" + tx_fname

                if metric == 'score':
                    info_file = open(info_file_filepath, 'r')
                    for line in info_file.readlines():
                        if metric_str in line:
                            score = float(line[17:])
                            designpath_metric_tup_list.append((design_path, score))
                    info_file.close()

                elif metric == 'matched':
                    info_file = open(info_file_filepath, 'r')
                    for line in info_file.readlines():
                        if metric_str in line:
                            frag_match_count = int(line[30:])
                            designpath_metric_tup_list.append((design_path, frag_match_count))
                    info_file.close()

    designpath_metric_tup_list_sorted = sorted(designpath_metric_tup_list, key=lambda tup: tup[1], reverse=True)

    if not os.path.exists(outdir):
        os.makedirs(outdir)

    outfile = open(outdir + "/ranked_designs_%s.txt" % metric, 'w')
    for p, m in designpath_metric_tup_list_sorted:
        outfile.write("%s\t%s\n" % (str(p), str(m)))
    outfile.close()
```

**utils/PostProcessUtils.py**

NanohedraManualUtils.py

```
def print_usage():
    print '\033[32m' + '\033[1m' + "NANOHEDRA\n" + '\033[0m'
    print '\033[32m' + '\033[1m' + "Copyright 2020 Joshua Laniado and Todd O. Yeates\n" + '\033[0m'
    print ''
    print '\033[1m' + '\033[95m' + "USER MANUAL" + '\033[95m' + '\033[0m'
    print ''
    print '\033[1m' + "QUERY MODE" + '\033[0m'
    print "REQUIRED FLAG"
    print "-query: used to enter query mode"
    print ''
    print "SELECT FROM ONE OF THE FOLLOWING QUERY OPTIONS"
    print "-all_entries: show all symmetry combination materials (SCMs)"
    print "-query_combination: show all SCMs that can be constructed by combining the two specified point groups"
    print "-query_result: show all SCMs that display the point group, layer group or space group symmetry specified"
    print "-query_counterpart: show all SCMs that can be constructed with the specified point group"
    print "-dimension: show all zero-dimensional, two-dimensional or three-dimensional SCMs"
    print ''
    print '\033[1m' + "DOCKING MODE" + '\033[0m'
    print "REQUIRED FLAGS"
    print "-dock: used to enter docking mode"
    print "-entry: specify symmetry combination material entry number"
    print "-oligomer1: specify path to directory containing the input PDB file(s) for the LOWER symmetry oligomer"
    print "-oligomer2: specify path to directory containing the input PDB file(s) for the HIGHER symmetry oligomer"
    print "            this flag is only used when both oligomeric components do not obey the SAME point group
symmetry"
    print "            for SCMs where both oligomeric components obey the SAME point group symmetry only the -
oligomer1"
    print "            flag is used to specify a path to a single directory containing the input PDB file(s)"
    print "-outdir: specify project output directory"
    print ''
    print "OPTIONAL FLAGS"
    print "-rot_step1: PDB1 rotation sampling step in degrees [default value is 3 degrees]"
    print "-rot_step2: PDB2 rotation sampling step in degrees [default value is 3 degrees]"
    print "-output_uc: output central unit cell for 2D and 3D symmetry combination materials"
    print "-output_surrounding_uc: output surrounding unit cells for 2D and 3D symmetry combination materials"
    print "-output_exp_assembly: output expanded cage assembly"
    print "-min_matched: specify a minimum amount of unique high quality surface fragment matches [default value is 3
]"
    print "-init_match_type: Specify type i_j fragment pair type used for initial fragment matching."
    print "                  Default is helix_helix: 1_1. Other options are:"
    print "                  helix_strand, strand_helix and strand_strand: 1_2, 2_1 and 2_2 respectively."
    print ''
    print '\033[1m' + "POST PROCESSING MODE" + '\033[0m'
    print "REQUIRED FLAGS"
    print "-postprocess: used to enter post processing mode"
    print "-design_dir: specify path to project directory (i.e. path to output directory specified in DOCKING MODE)"
    print "-outdir: specify output directory for post processing output file(s)"
    print ''
    print "1) USE ONE OR BOTH OF THE FOLLOWING POST PROCESSING FILTERS"
    print "-min_matched: specify a minimum number of matched fragments"
    print "-min_score: specify a minimum score"
    print "2) OR USE THE FOLLOWING FLAG TO RANK DOCKED POSES"
    print "-rank: followed by 'score' to rank by score or 'matched' to rank by the number of unique surface fragment
matches"
    print ''
    print '\033[1m' + "EXAMPLES" + '\033[0m'
    print 'python nanohedra.py -query -all_entries'
    print 'python nanohedra.py -query -combination C3 D4'
    print 'python nanohedra.py -query -result F432'
    print 'python nanohedra.py -query -counterpart C5'
    print 'python nanohedra.py -query -dimension 3'
    print 'python nanohedra.py -dock -entry 54 -oligomer1 /home/user/C3oligomers -outdir /home/user/T33Project'
    print 'python nanohedra.py -dock -entry 67 -oligomer1 /home/user/C3oligomers -oligomer2 /home/user/D4oligomers -
outdir /home/user/P432Project'
    print 'python nanohedra.py -postprocess -design_dir /home/user/P432Project -min_score 20.0 -outdir /home/user/
P432Project/PostProcess'
    print 'python nanohedra.py -postprocess -design_dir /home/user/P432Project -min_matched 7 -min_score 20.0 -outdir
 /home/user/P432Project/PostProcess'
    print 'python nanohedra.py -postprocess -design_dir /home/user/P432Project -rank score -outdir /home/user/
P432Project/PostProcess'
    print ''
```

**utils/NanohedraManualUtils.py**

# NANOHEDRA MANUAL

```
NANOHEDRA

Copyright 2020 Joshua Laniado and Todd O. Yeates


USER MANUAL

QUERY MODE
REQUIRED FLAG
-query: used to enter query mode

SELECT FROM ONE OF THE FOLLOWING QUERY OPTIONS
-all_entries: show all symmetry combination materials (SCMs)
-query_combination: show all SCMs that can be constructed by combining the two specified point groups
-query_result: show all SCMs that display the point group, layer group or space group symmetry specified
-query_counterpart: show all SCMs that can be constructed with the specified point group
-dimension: show all zero-dimensional, two-dimensional or three-dimensional SCMs

DOCKING MODE
REQUIRED FLAGS
-dock: used to enter docking mode
-entry: specify symmetry combination material entry number
-oligomer1: specify path to directory containing the input PDB file(s) for the LOWER symmetry oligomer
-oligomer2: specify path to directory containing the input PDB file(s) for the HIGHER symmetry oligomer
            this flag is only used when both oligomeric components do not obey the SAME point group symmetry
            for SCMs where both oligomeric components obey the SAME point group symmetry only the -oligomer1
            flag is used to specify a path to a single directory containing the input PDB file(s)
-outdir: specify project output directory

OPTIONAL FLAGS
-rot_step1: PDB1 rotation sampling step in degrees [default value is 3 degrees]
-rot_step2: PDB2 rotation sampling step in degrees [default value is 3 degrees]
-output_uc: output central unit cell for 2D and 3D symmetry combination materials
-output_surrounding_uc: output surrounding unit cells for 2D and 3D symmetry combination materials
-output_exp_assembly: output expanded cage assembly
-min_matched: specify a minimum amount of unique high quality surface fragment matches [default value is 3]
-init_match_type: Specify type i_j fragment pair type used for initial fragment matching.
                  Default is helix_helix: 1_1. Other options are:
                  helix_strand, strand_helix and strand_strand: 1_2, 2_1 and 2_2 respectively.

POST PROCESSING MODE
REQUIRED FLAGS
-postprocess: used to enter post processing mode
-design_dir: specify path to project directory (i.e. path to output directory specified in DOCKING MODE)
-outdir: specify output directory for post processing output file(s)

1) USE ONE OR BOTH OF THE FOLLOWING POST PROCESSING FILTERS
-min_matched: specify a minimum number of matched fragments
-min_score: specify a minimum score
2) OR USE THE FOLLOWING FLAG TO RANK DOCKED POSES
-rank: followed by 'score' to rank by score or 'matched' to rank by the number of unique surface fragment matches

EXAMPLES
python nanohedra.py -query -all_entries
python nanohedra.py -query -combination C3 D4
python nanohedra.py -query -result F432
python nanohedra.py -query -counterpart C5
python nanohedra.py -query -dimension 3
python nanohedra.py -dock -entry 54 -oligomer1 /home/user/C3oligomers -outdir /home/user/T33Project
python nanohedra.py -dock -entry 67 -oligomer1 /home/user/C3oligomers -oligomer2 /home/user/D4oligomers -outdir /home/user/P432Project
python nanohedra.py -postprocess -design_dir /home/user/P432Project -min_score 20.0 -outdir /home/user/P432Project/PostProcess
python nanohedra.py -postprocess -design_dir /home/user/P432Project -min_matched 7 -min_score 20.0 -outdir /home/user/P432Project/PostProcess
python nanohedra.py -postprocess -design_dir /home/user/P432Project -rank score -outdir /home/user/P432Project/PostProcess
```

*fin*