University of BRISTOL

Elfin: An Algorithm for Computational Protein Design using a Protein LEGO Strategy

Joy Yeh

April 2017

Final year project thesis submitted in support of the degree of

Master of Engineering in Computer Science and Electronics

Department of Electrical & Electronic Engineering

University of Bristol

Abstract

The ability to design custom 3D structures at nanoscale is becoming increasingly important due to the emphasis on miniaturised devises and an interest in spatially organised enzymes. Proteins are an attractive material due to their precise nanoscale arrangements, their homogeneity, and the biological functions they may bear. However, atom-to-atom interactions within proteins are complex and have for decades limited scientists' ability to design larger and more structurally varied proteins.

In this work we present Elfin – the first application developed to automatically design target protein sequences based on user-defined 3D shape criteria. Elfin exploits the stability and interaction characteristics of repeat protein units and uses them as basic building blocks for creating larger proteins, similar to how LEGO works. This thesis explains the favourable physical properties of repeat proteins and describes Elfin's Genetic Algorithm that is looking to bring about a new level of design control and simplicity.

In addition, we ported Elfin to OpenMP4 to take advantage of the accelerator targeting capability. The optimised performance allowed Elfin to design massive single-chain, irregularly shaped proteins (>1000 residues) in a matter of minutes. In our benchmark experiments, 30 shapes were tested and reported. To our knowledge, no existing method has been able to design single-chain proteins with no internal symmetry that are half as large as our smallest test case, which has 1149 residues. Finally, analysis on benchmark results suggests that our Protein-LEGO design model is capable of generating remarkably accurate structures. Such structures can in turn shed light on repeat protein characteristics that only manifest when they are part of a larger entity, as opposed to being in an isolated environment.

Table of Contents

List of Figures

List of Listings

List of Tables

Acknowledgements

I would like to thank Dr Fabio Parmeggiani for his original proposal of the new protein design strategy, as well as his tireless support during countless two-hour meetings on the biology and initial algorithm aspects of this work. Without his enthusiasm, this project would not come to be.

My sincere gratitude is also with my supervisor, Professor Simon McIntosh-Smith, who imparted expertise on performance optimisation and helped me gain access to an abundance of computing resources that made this project possible. Special mention goes to Cray Inc., University of Bristol's Advanced Computing Research Centre, and BrisSynBio for providing their compute clusters for use in this project.

Finally, I extend my appreciation to TJ Brunette and David Baker from the University of Washington for giving us permission to use some of their (unpublished) repeat protein designs for the database that is essential to this project.

DECLARATION AND DISCLAIMER

Unless otherwise acknowledged, the content of this thesis is the original work of the author. None of the work in this thesis has been submitted by the author in support of an application for another degree or qualification at this or any other university or institute of learning.

The views in this document are those of the author and do not in any way represent those of the University.

The author confirms that the printed copy and electronic version of this thesis are identical.

Signed: _____

Dated: _____

1. Introduction

1.1. Proteins for Non-Biochemists

This section introduces minimal protein concepts required to understand the rest of this thesis.

Proteins are tiny machines that mediate molecular functions vital to living organisms. Each protein is defined by a sequence of amino acid residues, and is studied using several 3D representations. *Figure 1* depicts what is called a *repeat protein* – one that consists of multiple tandem sequence repeats [1], [2]. The 'cartoon' display captures a convenient structural abstraction rather than a photo-realistic view. The 'stick' view in (c) shows amino acid residues that make up the *backbone* and the *sidechains* of a DHR18 repeating unit, which is formed by two *repeats*, the minimum number usually observed in nature.

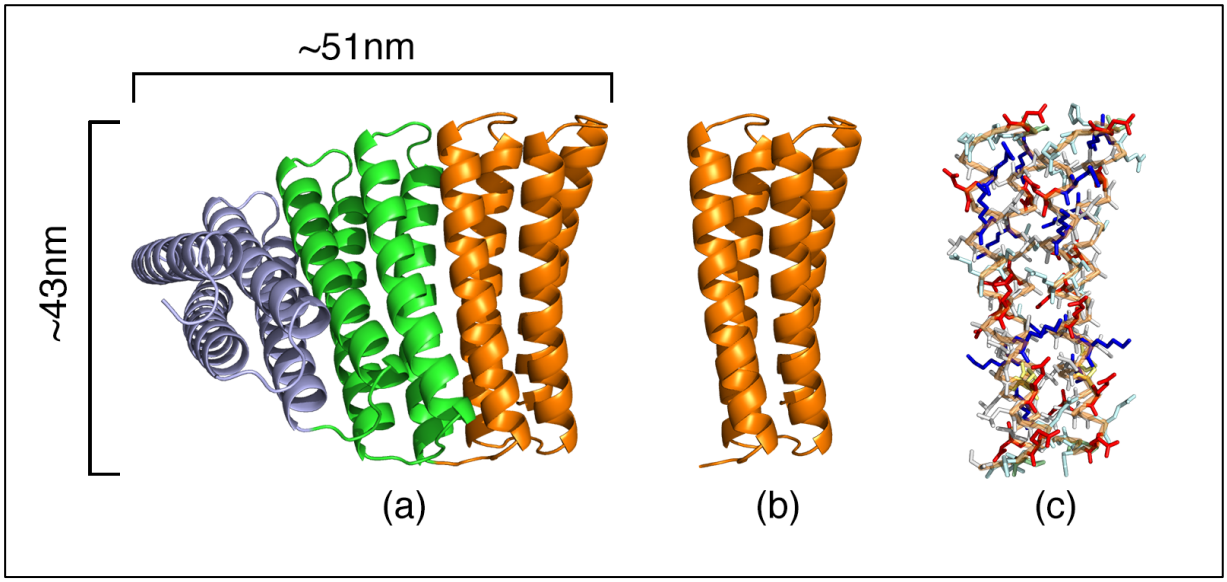


Figure 1. Scale and anatomy of the DHR18 repeat protein. (a): Three pairs of DHR18 'repeats'. (b): A 'repeat unit' formed by a two DHR18 repeats. (c): The same repeat unit rendered as amino acid sticks, with its backbone coloured in orange.

A protein can be made of tens to hundreds, sometimes thousands of amino acid residues. Each residue is a slot for one of the twenty natural amino acids. Every residue forms part of the backbone and one of the sidechains, so it is not wrong to think of them as trees. Together, all of the amino acid *conformations* determine how the protein folds. *Folding* refers the process of a protein creeping from its initial extended form to its folded, low energy form, whether happening in nature or *in silico*. Computationally simulating the folding physics of an amino acid sequence is called *structure*

prediction. It is an iterative optimisation process and a rigorous fold can cost considerable time, especially for large proteins.

Repeat proteins are of particular interest in this work because of their stability and interaction characteristics. There are many types of repeat proteins with different repeat sequence. These repeats always interface (i.e. form a peptide bond) with replicas of themselves, while some artificially designed versions are capable of interfacing with repeats from a different repeat protein [3]. A natural repeat protein can bear variations between its repeats, whereas in this work we focus on artificially designed and experimentally verified repeat proteins that have identical basic units [4]. There are no reports, to our knowledge, of single repeats being able to fold in the correct conformation. Therefore two repeats are usually considered the minimal repeat 'unit' [5], [6]. Prior research has found that these repeat units only interact with their immediate neighbours [7], so it is theoretically plausible to construct larger but still relatively stable proteins using these units as building modules.

1.2. Protein Design

Protein design is the reverse problem of structure prediction – it searches for an amino acid sequence that folds into a certain desired 3D arrangement. Being able to create new proteins not found in nature unlocks deeper insights into how proteins fold, and how to engineer proteins. There are two main categories of protein design: functional and structural. The former aims to create proteins that bear new functions or can be activated under a controlled range of conditions. The latter focuses on generating proteins structures that can host a range of functions at specific sites. The ability to achieve these goals can translate to exciting potentials in medical sciences and industry.

This project focuses on the design of large proteins that can hold multiple functional parts, such as enzymes, in specific 3D arrangements. One particular motivation for creating these protein 'frames' is to provide behaviour control of cells from bacteria, animal, or plants through cell signals. The spatial organisation of signalling molecules affects cell response and its decision to grow, replica, or develop into a new type of cell [8]–[10]. However, there are only very few tools that allow efficient control of 2D display of signalling molecules and for investigating how signals' spatial arrangement affect cell behaviour [11], [12]. New tools developed in this direction might have far-reaching implications, particularly in stem cell research and regenerative medicine, allowing production of blood and other biological tissues for transplants in an efficient and affordable way.

In order to freely explore nanoscale spatial configurations, we sought to design large protein structures not bound by symmetry constrains. We propose to experiment with using repeat protein units as basic building blocks. Multiple of these different units can be assembled in simulation to form an overall

custom 3D shape defined by the user. *Figure 2* serves to illustrate the simplified workflows of existing protein design methods and the difference in our 'Protein LEGO' strategy.

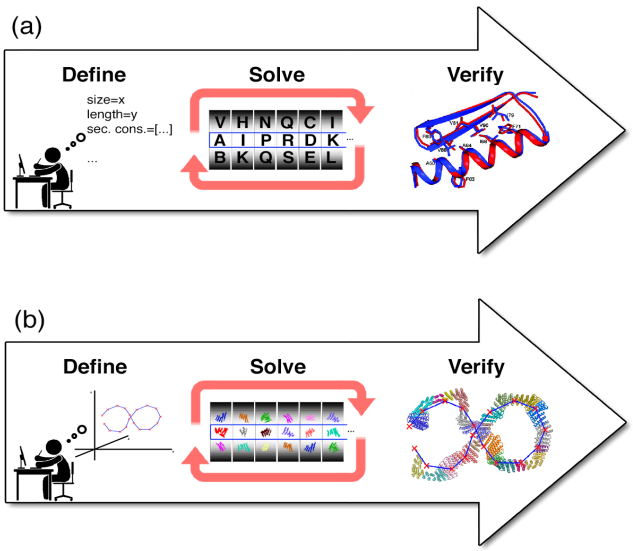


Figure 2. High-level protein design stages. (a): Workflow of traditional protein design methods that search the amino acid sequence space for a desired fold. The verification image is the Top7 design taken from [1]. (b): Our Protein LEGO building strategy, whereby repeat protein units are used as building blocks as opposed to single residues or peptides.

Protein design begins with the definition of design criteria. The picture of this stage in *Figure 2-a* conveys that existing design methods are limited to spatially imprecise parameters such as protein size and secondary structure contents (with some exceptions; more on this in Section 2.1). In contrast, our work on Elfin allows us to generate proteins based on shape criteria.

A computational search then follows in order to solve for amino acid sequences that are compatible with the desired shape (according to physics simulations or abstractions thereof). At this stage, our search method differs significantly from other search methods in that we use radically different, large (80~200 residues) building blocks with well-defined sequences. These repeat protein modules are subject to different but favourable physical rules that requires a new search algorithm to take advantage of.

In the last stage of the design workflow, structure verification is performed on full atom representations of the design. It is possible that the relaxed (sidechain-repacked and energy-minimised; [13]) final design structure deviates substantially from the intended fold of the designed protein. By confirming that atom coordinates in the two structures are highly similar, protein designers can filter out low quality structures before putting them into synthesis.

1.3. Contributions

In this work we present an automatic protein design application for large and massive proteins. The application, called Elfin, is equipped with a modular assembly algorithm that constructs target proteins to exhibit shapes required by the design specification. This style of design specification is simpler, less restrictive, and geometrically more specific than those employed by existing methods [14]–[19]. In our benchmark tests, massive proteins of size ranging from one to five thousand residues have been designed to remarkable accuracies. We believe these are the largest single chain, asymmetric *de novo* proteins[1] ever designed to date. It would be computationally intractable to design the same structures using existing approaches.

The main contribution of this thesis is our description of the Elfin software pipeline. Elfin is open source and available online at https://github.com/joy13975/elfin. To the best of our knowledge, it is the first application to offer an intuitive spatial control for protein design. We also present 30 massive *de novo* proteins designed using Elfin, with independent protein relaxation verifications. In fact, Elfin is a generic rigid body construction application, not just limited to repeat protein units. This means with some modification, Elfin can be repurposed for different construction problems such as LEGO blocks. While this thesis focuses on the design and implementation aspects of Elfin's algorithms, a more Biology-focussed journal paper is being prepared (Abstract in Appendix B).

In order to accommodate the need for a tight design-redesign cycle and as an effort to push protein design towards interactive speeds, Elfin was optimised for performance and ported to OpenMP4 with GPU targeting capability. An account of the steps involved in optimising Elfin is our second contribution.

The structure of this thesis is as follows. A review of past protein design methods and computer software is given in Section 2.1. Theoretical backing and empirical evidence for the use of repeat proteins is discussed in Section 2.2. Section 3 describes the Elfin application pipeline and its core algorithms responsible for assembling repeat protein units. Lastly, output structure accuracy and performance benchmarks are evaluated before this thesis concludes.

[1] Large *multi-chain* and *symmetric* proteins have been designed before [28]–[30], but these methods create regularly shaped designs that adhere to a parametric description.

2. Background

2.1. Protein Design Approaches

This section gives a review of existing methods and computer software for designing proteins and related nanostructures.

Earliest work on protein engineering dates back to 1975 by Gutte and colleagues [20], who used the 70-residue RNase S protein as a template for experimenting with deleting and substituting residues. A decade later, Ponder and Richards proposed the fixed backbone design method with statistical sidechain conformation preferences, or 'rotamers', in order to narrow down the vast amino acid sequence search space [21]. Their proposal led to considerable computational savings and had profound impact in subsequent search algorithms, notably [22] and [23]. Several pre-2000 computational methods were reviewed by Desjarlais and Clarke in [24].

Fixed backbone methods imply limited structural freedom and dependence upon natural protein templates. The first non-fixed backbone, fully *de novo* designed protein to be experimentally verified was the 93-residue 'Top7', created by Baker and colleagues in 2003 [14]. In their work, a general design procedure was developed, which iterates between Monte Carlo sequence optimisation and structure prediction. Their design process employed a secondary structure schematic that specifies which residue should belong to which secondary structure and what type. This work, together with several others ([14], [25]–[27]), exploited multi-residue (peptide) sampling to reduce the conformational space search. Techniques for designing *de novo* proteins might become increasingly coarse-grained, because in order to accommodate the design of larger proteins the explosion of search space must be controlled. As a rough but conservative example, a 100-residue protein has a sequence search space of 20^{100} when considering the twenty natural amino acids. This is without taking into account the conformations (or rotamers) that each residue may assume.

Other classes of structural design algorithms also exist, such as those based on parametric helices and coiled-coils [28]–[30], and kinematic closure methods [31], [32]. Parametric methods are able to produce single and multi-chain proteins[2] (100~200 residues) that are ensembles of repeating units embodying the geometric properties described by the parametric equations. On the other hand, kinematic closure algorithms are able to link two peptides together (i.e. perform 'loop-closure') by

[2] Chains denote physically separate amino acid sequences that are very close in space and are held together by atomic forces rather than chemically connected via a peptide bond.

considering atom-atom interactions in a relatively local scope. Kinematic closure has been used to design smaller (<50 residue) proteins by applying multiple closures within the chain [31].

More recent computational protein design algorithms were summarised by Gainza *et al* [33]. Despite the many algorithmic improvements, single-chain *de novo* designs beyond four hundred residues remain an unrelenting challenge with currently available techniques. The reported methods are either intended for small to medium sized proteins, or specialised for multi-chain protein complexes constructed based on symmetric components [34]. The sheer number of residues in designing a large (>300 residues) protein makes current algorithms prohibitively costly in terms of time. In this work, we are interested in structures that are larger than one thousand residues, the size at which the designed proteins may begin to hold multiple functional parts with precise positional control. Based on this, we justify a need for an even more coarse-grained design approach – by using repeat protein units and exploiting the locality of their force interactions [4], [7]. Further discussion on repeat proteins is in the next section.

At the time of writing, a number of Computer-Aided Design (CAD) software packages are available for designing nanostructure that reflect a user-defined 3D shape. DNA origami can be created using DAEDALUS[3] [35] (for regular, closed-surface shapes) or Cadnano[4] [36] (for arbitrary but manually 'carved' designs). CoCoPOD[5] is a tool similar to DAEDALUS, but creates polyhedral meshes using coiled-coil pillars. Others include InteractiveRosetta [37] and Maestro[6] by Schrödinger LLC. Having listed the above applications, it is important to differentiate automatic design from graphical modelling. To our knowledge, only DAEDALUS and CoCoPOD are capable of *automatically* computing the 'recipe' for building a given shape design. The rest of the mentioned applications require user's manual input, i.e. to craft a desired shape out of DNA strands or by manoeuvring amino acid residues, subject to movement constraints while doing so. To summarise, we were unable to find existing software or algorithm that is capable of automatically and efficiently designing large protein nanostructures (>300 residues), especially not one that takes a set of 3D coordinates as input specification. This makes Elfin the first protein CAD application of its kind.

[3] http://daedalus-dna-origami.org/
[4] http://cadnano.org/index.html
[5] https://github.com/NIC-SBI/protein_origami
[6] https://www.schrodinger.com/maestro

2.2. Repeat Proteins

This section discusses the properties of repeat proteins and why they are promising building blocks for constructing larger proteins.

Repeat proteins are composed of multiple repeating tertiary structure motifs [38] or 'tandem repeats', and are commonly found in nature. They play a key role in essential biological processes by supporting functions such as macromolecular binding, molecular recognition, enzymes, and signalling [2]. Structure-wise, studies have reported highly diverse overall architectures in repeat proteins despite the name 'repeat' [39]–[43]. Typical repeat proteins have two termini, and each interacts with the opposite terminus of another replica of the same repeat. It is noteworthy that the interfaces are directional.

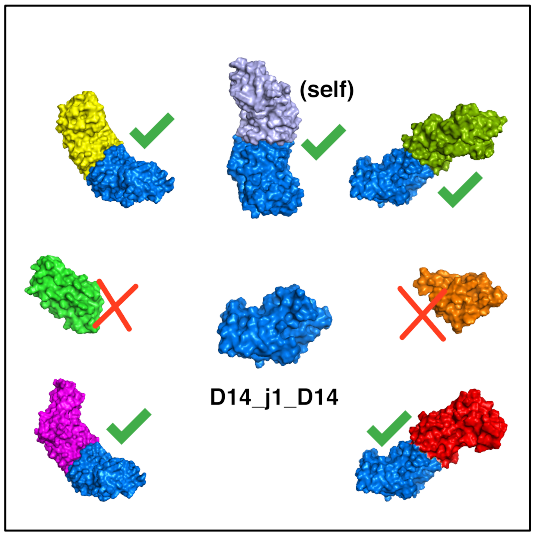


Figure 3. Illustration of D14_j1_D14 interface compatibility. (self): A replica of D14_j_D14, coloured differently to indicate that they are separate entities.

A repeat protein code-named D14_j1_D14 is shown in *Figure 3*, again designed by Brunette and Baker in their not-yet published experiment. This is a 'surface' render of several units that are coloured differently to denote that they are units belonging to different repeat proteins. The figure shows that while the D14_j1_D14 unit can interface with four repeat units from different repeat proteins (and also with itself), it does not interface with two other repeat units. This is by no means an exhaustive list of D14_j1_D14-compatible units, but rather an illustration of repeat unit relationship and the fact some repeats can be 'picky'.

Park *et al* showed that repeat proteins can be made to form specific curvatures by engineering the interfaces used in joining repeat units [3]. This suggests that by manipulating and combining different repeat protein units, shape-specific protein design is theoretically possible. Repeat proteins were also found to physically interact only with their immediate neighbours [4], [7], [1]. If this is true, then a repeat protein-based construction algorithm may be spared from having to compute pair-wise interactions.

These attractive characteristics would not be as hopeful if it were difficult to produce repeat proteins or to design new repeat proteins. Fortunately, prior efforts have already succeeded in engineering and

re-purposing natural repeat proteins for specific molecular recognition and scaffolding functions [44]–[46]. Baker and co-workers took it a step further and designed 83 *de novo* repeat proteins with very low similarity to any known repeat proteins [47]–[50]. Out of the 83 *designed helical repeats* (DHRs), 53 were experimentally verified as consistent with design models and characterised as highly stable. These studies show that there is active research on generating new repeat proteins, and that procedures have been developed that could design fully *de novo* repeats with an excellent verification success rate.

In their unpublished work, Brunette and Baker designed and expressed DHRs that host different interfaces on each terminus. These DHRs were named 'junctions' (e.g. D14_j1_D14 in *Figure 3*) because they allow different repeat protein units to bind to their termini. They then characterised both the DHRs and the junctions using Small Angle X-ray Scattering. The experimental data indicated that in solution the proteins match the design models and the crystal structures in the case for which they were determined. Thus, we can hypothesise that these DHRs are rigid bodies.

The 'pickiness' and 'short-sightedness' of repeat proteins help reduce the algorithmic complexity in the construction of a large protein. Their rigidity would also allow construction using some form of reference frame cascading (jump to *Figure 5*). We already have database of over a hundred repeat protein pairs, and it is only growing in variety. For these reasons, we believe that repeat proteins are promising candidates for use as basic building blocks for larger proteins.

3. Methodology

3.1. Overview

This section is dedicated to describing the Elfin protein design pipeline (*Figure 4*). Elfin is a suite of Python scripts with a core Genetic Algorithm written in C++. The main design objectives of Elfin are:

a) Take as input a sequence of 3D points defining a line that the target protein should exhibit as its overall architecture.
b) Produce a single-chain protein that folds into the 3D design points.
c) Run efficiently with unit tested code so that it is practical for repeated use in future research.
d) Be as accessible as possible, without targeting any specific hardware or operating system.

For specifying 3D points to express shape criteria, we developed a Matlab script that allows the user to plot points and scale the shape appropriately. Designing a GUI is outside the scope of this project and the script we have developed has proven sufficient in generating results needed for this work.

In addition to shape specification, Elfin also requires a database of repeat protein pairs and units that provides building blocks for the output protein sequence. For this project, we use a database prepared by Dr Fabio Parmeggiani at the University of Bristol, with junctions acquired from Brunette and Baker with their permission. This database is not yet published, and will consist of 35 single building blocks (derived from 12 ‘pure’ DHRs and 23 junctions) and 143 building block pairs covering known connections.

The following subsections walk through the manipulation of data in (roughly) chronological order, from the bottom right-hand-side to the top left-hand-side of design cycle in *Figure 4*.

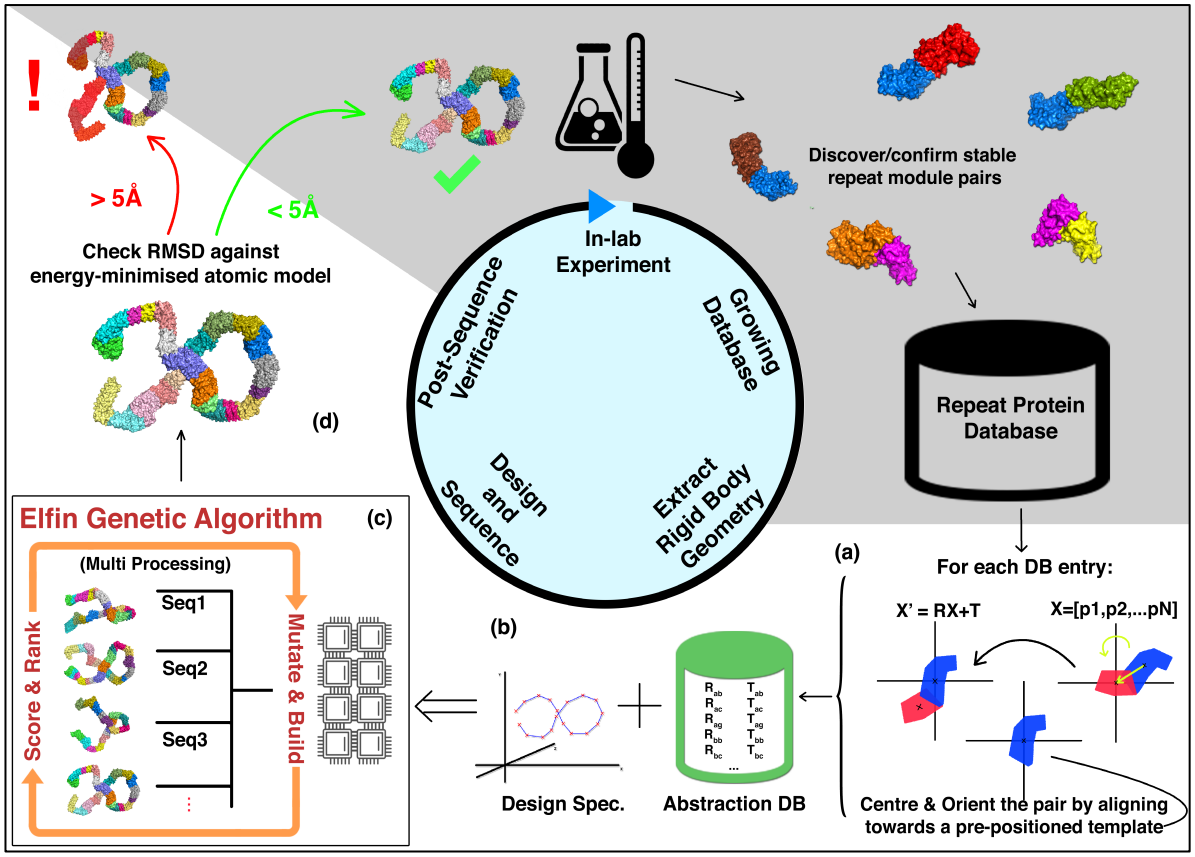


Figure 4. An overview of the protein design cycle. The grey sector denotes part of the design cycle not covered by the Elfin pipeline. Going clockwise, the cycle iterates in the order of: laboratory experiment, database growth, abstraction and sequence design, and finally structure verification before the output is synthesised in another experiment. Icon attributions[7].

3.2. Pre-processing

The repeat protein database is a set of PDB [51] files, each containing atom types and coordinates of either a repeat protein pair ('pair' hereafter) or a single repeat unit ('single' or 'module' hereafter). All pairs were first chain-merged and loop-closed so as to correctly represent each as pair a single physical entity. Each PDB is then cleaned of artefacts and relaxed using tools from the Rosetta Software Suite [52]. At this stage, the average atomic Root-Mean-Square Deviation (RMSD) of database PDBs

[7] Designer icon: clipartfest.com; Chemistry icon: freeiconspng.com; All other icons: iconfinder.com.

before and after relaxation was 0.77Å, which is very small and expectedly so. These steps ensure that Elfin uses building blocks that are correctly represented and that the resultant protein sequence is less likely to break free from the intended formation due to loose, high-energy module structures.

The next step is to reduce each pair from their atom coordinates to a very compact 6-float transformation descriptor. This array describes exactly one rotation (3 angles) and one translation (x, y, z) for re-aligning a protein sequence during the building process, which is discussed Section 3.3. Elfin uses the following routine to produce the *abstraction database*:

```
def createAbstractionDB(singles, pairs):
      for single in singles:
            moveToOrigin(single)

      txDescriptors = []
      for pair in db_pairs:
            pair.transformByAlignment(pair.A, singles[pair.names.A])
            rot = pair.getRotationTo(singles[pair.names.B])
            tran = pair.getTranslationTo(pair.B)
            newDescriptor = {rot, tran}
            txDescriptors.append(newDescriptor)

      return txDescriptors
```

Listing 1. Python-like pseudocode of the geometry abstraction routine.

The routine in *Listing 2* sets up a 3D reference frames that encode geometry assumptions inherent in the protein pairs' organization (a simplified 2D illustration can be found in *Figure 4-b*). The first loop moves every module to the origin, making each single an 'alignment template' for subsequent pair transformations. In the second loop, each pair is aligned to the centred version of the first single (arbitrarily named 'A') that is part of the pair. Note that in the database, singles and pairs are separate objects and that each pair is not exactly equivalent to the docked (non peptide-bonded) combination of two singles, because the structure was energy-minimised in the presence of its partner unit. Each pair, however, does contain all the residues (and hence atoms) present in the two singles that it is made of, thus allowing alignment through superimposition.

The second loop then extracts and stores the transformation required to move the aligned pair to the second single (arbitrarily named 'B'). Through this routine, we obtain all the information needed to transform each aligned pair such that before the transformation, component A of the pair is centred, and after the transformation, component B of the pair is centred. This constitutes the "Extract Rigid Body Geometry" stage of the Elfin process in *Figure 4*.

3.3. Core Algorithms

The core algorithms compute a sequence of single modules that best fit the shape specification provided by the user. Regardless of which meta-heuristic is to be used, two subroutines are required:

A way to build a protein sequence: Using the transformation descriptors from the abstraction database, we developed a "Place-and-Push' construction process (*Figure 5*) similar to how Ribosomes synthesise proteins. Ribosomes are biological workbenches that translate mRNA protein 'recipes' into proteins, working sequentially and only on a single site at a time. It can be likened to the way tape computers read punched tapes, except a structure is being made on the other end as each package of data gets processed.

Suppose a module sequence is known, the sequence built so far in the current step is moved to the origin. Then, the next module is added. This process implicitly cascades the reference frames of each pair, and is repeated until the end of the sequence. The geometry of the overall shape consists of the centres of mass (CoM) of the constituent modules. Collision checks are performed by asserting that the Carbon-Alpha gyradii[8] are not violated when placing the next module.

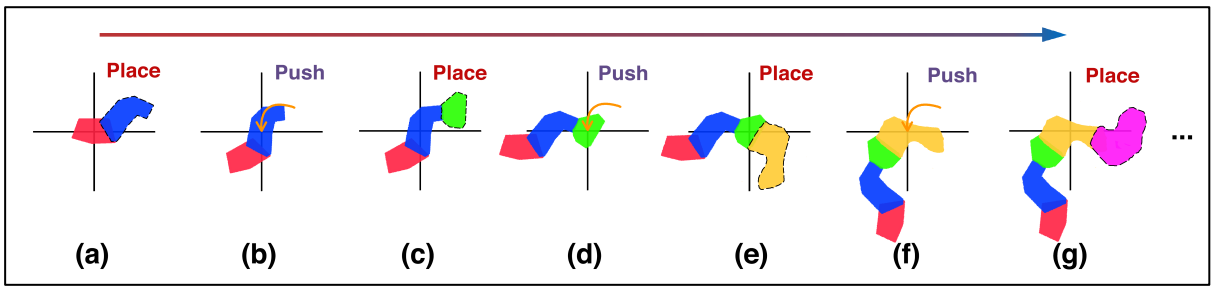


Figure 5. 3D illustration of the Place-and-Push sequence building process. In each step from left to right, the protein body built so far (just a vector of CoMs) gets transformed such that the newly added single is aligned to its alignment template. This alignment allows the precise placement of the next module, and so on.

A way to score a protein sequence: A measure of dissimilarity between two sets of 3D points is needed to tell a better module sequence from a worse one. This is in fact a partial Procrustes shape analysis problem [53], which we solved using the Kabsch absolute orientation algorithm [54]. In our implementation of the Kabsch algorithm, an RMSD score is computed after superimposing two sets of

[8] Gyradii is the plural form of gyradius, which stands for Radius of Gyration. This is the average distance from all Carbon-Alpha atoms to the CoM of a protein. Elfin also supports other measures, such as average distance of all heavy atoms.

points. However, there was one complication: it is impossible to guess how many modules will form the best sequence for a given design, since the solution for a real design is unknown. In other words, if the user provides a hand-drawn 3D shape, it is not guaranteed that each point in the specification can be met exactly by constructing a protein sequence. A very simplified analogy in LEGO terms would be that one simply cannot build a triangle using just two-by-eight rectangular bricks, but those who do not understand this may try for a long time (except repeat proteins are not so dully shaped). The implication is that Elfin must allow for exploration of sequences that deviate from the expected length (in number of modules). Furthermore, since the Kabsch algorithm cannot deal with shapes containing different number of points, we developed a proportional re-sampling subroutine to equalise the number of points defining the two shapes in question. In short, Elfin up- or down-samples the candidate shape according to the proportion of length at which each point appeared in the specification. This is reasonable because we expect a fit sequence to have a point at X% into the shape that is close to point Y, for all points in the specification.

Having devised the above two subroutines, a meta-heuristic is due. Before diving into a myriad of machine-learning techniques, we first implemented a Greedy algorithm to test whether there is even a need to venture beyond the simplest. The Greedy algorithm is the simplest next to trying random sequences or using brute-force. At each step, the algorithm selects the module that would produce a partial shape that best fits the specified shape. To test the algorithm, we generated ten benchmark problems by building random sequences. Not surprisingly, the Greedy algorithm was able to find the correct solutions for these problems in a very short amount of time. However, when real design problems were provided as input (hand-drawn shapes using our Matlab plotting script), the Greedy implementation consistently produced poor solutions that clearly failed to resemble the desired shape. Upon inspection, it was clear that due to its inability to backtrack, the Greedy algorithm often ran into module 'dead-ends'. This is because module compatibility is directional, and some modules allow only one type of following module. Once a Greedy algorithm descends into one of these dead-end modules, it becomes deprived of the variety of modules required for producing a quality solution.

In addition to a large search space (4.6^N for an N-module design using the current database[9]), the evidence that the Greedy approach could not provide the needed quality further justified the use of advanced meta-heuristics. In attempting to choose a different algorithm, we observed that protein

[9] The expression 4.6^N was computed by summing the number of paths found in each N-powered adjacency matrix.

sequences were already structurally similar to that of chromosomes in terms of data in memory and source code. The way in which Natural evolution applies modifications to chromosome is also applicable to protein module sequences. Evolutionary algorithms such as Genetic Algorithms (GA) have been successfully applied to both protein design and structure prediction before, albeit at a different scale and using different subroutines to construct the protein [22], [55]–[58]. Such algorithms simulate evolutionary pressure on a population of solutions that gradually mutate to become more fit.

Another compelling algorithm we considered was the Dead-End Elimination (DEE) method for finding minimum energy sidechains [59]. DEE led computational breakthroughs in the late 1990s [23], [60] and early 2000s, and was shown to still be the dominant choice for many protein design applications [61]. The basic idea of DEE is that once a sidechain rotamer can be shown to not form part of the global minimum sequence at that specific position, it can then be relieved and not be searched further (at that specific position in the sequence). Unfortunately, suspending a module from being selected at a certain position such as done by DEE has different implications in our application. Each amino acid is compatible with any other amino acid, but that is not the case for the building blocks in Elfin. If we remove one module from being selected at any position, it does not only mean that it cannot occupy that position, but it also affects subsequent modules due to their non universal compatibility relationship. However, eliminating highly improbable or straight impossible modules based on their geometry is a complementing idea with Elfin. Although we may not use DEE for this application, we believe a GA implementation with hints of DEE philosophy may prove effective.

Elfin's core GA is depicted in *Figure 4-c*. Stages of each GA generation are shown in *Listing 2*. In the first generation, all individuals are created by randomly generated module sequences, which could differ in length but are restricted to a user-defined distance from the expected sequence length. During subsequent generations, the GA applies three mutagenesis operators to modify protein sequences (i.e. individuals of the GA population), which are illustrated in *Figure 6*. Again, these operators may change the sequence length of an individual, the extent to which is limited by a program parameter. After evolution, a fitness score is assigned to each individual using the Kabsch RMSD algorithm. The GA then ranks its population according to ascending fitness scores (lower is better). Lastly, fit and provably diverse individuals are selected to 'survive', while the rest undergo inheritance and mutations again, thereby exerting evolutionary pressure.

```
def GeneticAlgorithm(population, maxIters):
      for i in 1 … maxIters:
            evolve(population)
            score(population)
            rank(population)
            select(population)
```

Listing 2. Python-like pseudocode of Elfin's GA stages. Evolve() applies the three different mutagenesis operators on each individuals, with the type of operator determined by rolling a dice against user-defined probabilities. Individuals that do not survive a generation would inherit a random parent sequence, then mutate randomly. Score() simply applies the Kabsch score function on each individual. Rank() sorts the population by fitness, and Select() enforces population diversity by ensuring that no two identical individuals survive in the same generation.

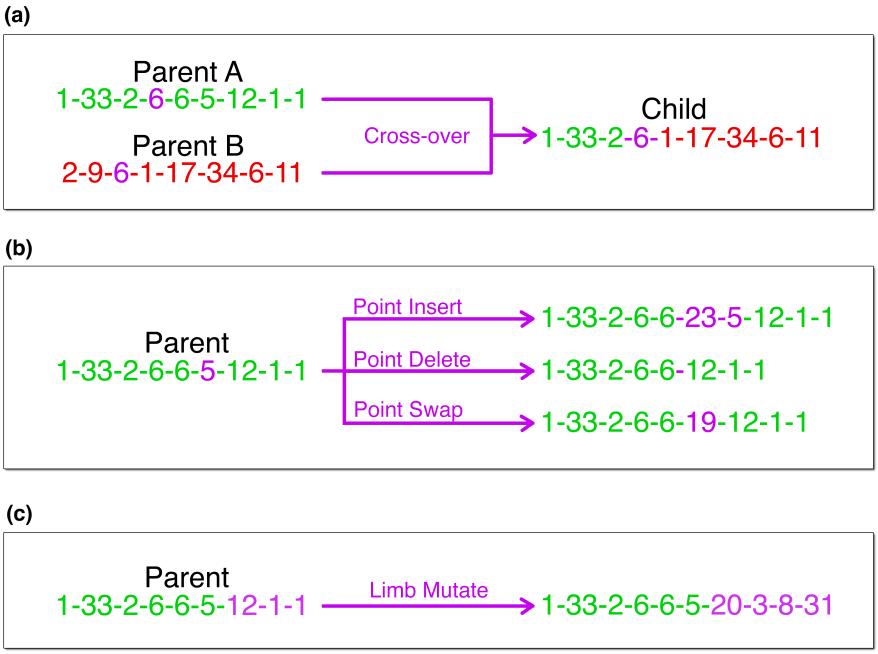


Figure 6. Demonstration of Elfin's three mutagenesis operators.The numbers symbolise unique module identifiers. (a): Crossover creates a new child by combining two parent sequences at some compatible point. (b): Point mutation is one of: insertion, deletion, or swap, performed on one gene. (c): Limb mutation severs all genes on one randomly chosen side of a randomly chosen gene position. The limb is then randomly regenerated.

In addition to the population size and maximum iteration settings, Elfin has the following tuneable parameters that affect the solution quality and its execution time:

- **Average Pair CoM Distance**: the average CoMs distance of all database pairs, used to compute an expected sequence length for real design inputs. This is value is found by analysing the abstraction database (~3Å) and would be fixed unless the database changes.
- **Chromosome Length Deviation**: the number of modules from the expected sequence length that an individual is allowed to deviate.
- **Survival Rate**: the ratio of population to survive each generation.
- **Crossover Rate**: the ratio of non-survivors to undergo crossover.
- **Point Mutate Rate**: the ratio of non-survivors to undergo point mutation.
- **Limb Mutate Rate**: the ratio of non-survivors to undergo limb mutation.
- **Score Stop Threshold**: the score below which the GA should consider the solution 'good enough' and halt the search.
- **Maximum Stagnant Generations**: the number of non-improving iterations after which the GA must give up.

Two grid searches were conducted in order to determine optimal parameters for Elfin. In the first search, a wide spread of discrete values were tested (Appendix C) on benchmarks and on real designs (more on this in Section 3.4). The Pareto Front taking solution score (lower is better) and execution time (shorter is better) as objectives was identified to guide a second search that covered a narrower spread of discrete values (Appendix D). The configuration that yielded the best objective metrics can be found in Appendix E. After two searches the configuration did not seem to shift from the first Pareto Front, leading us to believe the values we found make for a reasonably good setting.

It is important to recognise that the optimal configuration can change as Elfin gets updated with new features and algorithm improves in the future. Our optimal configuration is likely to be different to another users' due to differences in hardware, affordable compute time, inputs used, and minimum desired solution quality. Therefore, it is recommended that the user perform their own grid search if our configuration does not produce satisfactory results (or execution times).

3.4. Post-processing and Verification

The only post-processing in Elfin is the conversion of module identifier sequences to atomic representations (PDBs). This is done using a Python script that parses the output solution and consults the repeat protein database in a way that appropriately samples module interface residues.

In order to ensure and maintain Elfin's correctness, unit tests have been put in place and used throughout development. This was especially important for the matrix mathematics portion of the code that is difficult to check with manual inputs. Elfin also includes a random number seeding option that can be used to force identical output for the same input. This add consistency to debugging and measuring average number of iterations before convergence.

The proposed protein design model was tested using 20 random sequence benchmarks and 10 hand-drawn 3D shapes. The random sequence benchmarks were generated by extracting CoMs from randomly 'grown' module sequences. Each CoM sequence constitutes an input design specification to Elfin. If the algorithm we have described is effective at solving the building problem and the sequence search, Elfin should give solutions identical to the module identifier from the randomly generated benchmarks. The other hand-drawn shapes were used to learn about Elfin's effectiveness at solving real designs. We introduced seven written letters from the word 'BRISTOL', and three custom twisted shapes (horns, spiral, and infinity sign). Some of these are simple geometries, such as 'I', 'L' and 'S', that Elfin should succeed at designing. Mean while, others with sharper turns, closed loops, and convoluted 3D geometry such as 'B', 'R', 'T', and the twisted shapes, test the bending and scaling limits of the current repeat protein database. As a matter of comparison, the smallest 10-module random sequence we use to test Elfin's designing capability has 1460 residues (recall that we know the solution because it was a generated case). To our knowledge, no single-chain protein in the field of *de novo* protein design has exceeded 400 residues.

After running Elfin through all the benchmarks, the post-processed PDBs underwent structure relaxation using Rosetta. This is process that iteratively searches for a lower energy (i.e. more compact and stable) state of a protein's atoms using sidechain repacking and energy-minimisation with repulsion energy ramping [13]. Notably, relaxation is more rigorous than just energy-minimising the structure and can introduce larger deviations in its output. Finally, atom coordinates of each PDB before and after relaxation were compared to generate RMSD scores. We considered two types of RMSDs: the global RMSD, and the *windowed* RMSD(s). The former describes the overall structure similarity, while the latter is able to locate the problematic sections of the design. The maximum of the windowed RMSDs for a given design is also a measure of the *worst local distortion*, which is particularly useful in analysing problematic sections of elongated bodies. We compute the RMSD by

using a 300-residue window with 50% overlap, which slides across the entire result chain rather than on a per module basis.

3.5. Optimisations

There are several motivations for optimising Elfin for performance. Firstly, while the number of protein module combinations in a sequence is not as large as that of the amino acid sequence of the same protein size (e.g. an Elfin design of 40 modules can exceed 10,000 amino acids), it is still astronomical. At the time of writing, the repeat protein database is at a size of 143 pairs and 35 singles. With this database Elfin's search space is $\sim O(4.6^N)$ for an N-module sequence search. It is important to note that the Protein LEGO sequence search should not be simply compared against single amino acid or peptide fragment searches, because each aims to design at a different scale, uses different building blocks, and satisfies different design criteria. Furthermore, the repeat protein database will only increase in size as more repeat protein designs get experimentally verified in the future. Each new repeat protein unit increases the base number of the search space size, which is why we shall not dismiss the current search space as 'small enough'.

Secondly, of the many existing CAD applications, proteins have not seen one that offers much design freedom while being able to give feedback at interactive speeds. Recall the nanostructure CAD applications discussed in Section 2.1: these are either graphical modelling tools requiring the user the craft out a structure under to movement constraints, or automatic design algorithms that could produce only symmetrical complexes of DNAs or coiled-coils. If Elfin could produce quality solutions in a short amount of time, perhaps it may be the first interactive protein CAD application that designs for freeform 3D shapes.

Lastly but not least, when Elfin is used in practice it is expected that for a specific shape, the user might need to run Elfin multiple times in series due to unsatisfactory solutions. Note that the cause of insufficient solution quality lies not in the algorithm Elfin uses, but in the fact that discrete protein sequences can only cover a finite number of discrete shapes constrained by structure collision and compatibility (recall the LEGO triangle analogy). With real design problems such as those drawn by hand, chances are that one corner is too sharp or one spiral is too tight for the protein modules to assemble in space. If Elfin took a long time to solve a typical design problem, then it would fail to meet the design goals we set forth in Section 3.1. On a related note, there are applications for massively parallel designs. The user may generate a large list of possible poses at which to hold enzymes or similar activities, because without already knowing the final structure, it is difficult to 'just know' which pose is optimal. These can be computationally converted into 3D lines that are then fed through Elfin to design in compute clusters.

The above motivations led us to introduce algorithm and code optimisations to Elfin. To begin with, Elfin was written from scratch using C++ without using external libraries (other than the C++11 standard library[10] and a JSON parser[11] for file IO). The reason for not using third party codes was to preserve control of the source code and allow significant (or micro) architectural changes when and if performance optimisations call for it. There were many algorithm changes that led to faster serial runtimes, but since Elfin is a newly developed application, less significant speed-ups were not tracked. Here we shall highlight three that made major impacts:

1) **Removal of crossover compatibility pre-computation**. Initially it was thought that pre-computing pair-wise crossover compatibility would help save time as crossovers were assumed to be very common. This was in fact a performance bottleneck because the 'pre-computation' took place in every iteration, and there was not nearly enough crossovers per generation to justify computing all pair-wise crossover compatibilities. By removing this, some benchmarks saw speedups of as much as 2x.
2) **Limiting mutation tries**. During evolution, mutations can fail when the operator introduces collision into the sequence i.e. reaches module dead-ends. Before a trial limit was imposed, Elfin saw performance bottleneck in its evolution phase as some individuals required many more mutation tries before a successful mutation was applied. This was partly the problem of stochastic mutation, but it is a justification to remove the stochastic characteristic of the GA. By forcing mutations to give up after a certain number of tries, Elfin's runtime was improved by around 30% on average.
3) **Survivor diversity enforcement**. This feature was added when it was realised that Elfin tended to converge to many identical survivors. Suffering from a lack of diversity, Elfin needed a large number of iterations (usually well over 100) to produce satisfactory solutions. By implementing a fast Cyclical Redundancy Check routine, we were able to differentiate each individual using just one long integer and ensure that no duplicates survive in each generation. This reduced the number of iterations Elfin took for solving benchmarks to below a hundred for the typical case, but increased computations per iteration due to the added CRC calls and duplicate exclusion step. Overall, the convergence times improved by roughly twice.

[10] https://isocpp.org/std/the-standard
[11] https://github.com/nlohmann/json

Other minor code changes such as minimising memory copying and overall footprint, inlining frequently called subroutines, and favouring primitive data structures over classes and strings, have made smaller speed improvements. Some of these were implemented from the beginning, so there was no base line to compare against. We believe that critical modifications applied to Elfin's GA and our performance-aware development has made our implementation of the algorithm reasonably efficient. Since Elfin is the first application of to lead in the case of large automatic protein design, we also expect many future changes and better algorithms to be proposed that improve upon the ones described herein. One of our design objects was to not over-engineer Elfin so that it will not be difficult to make changes later. Therefore we purposely did not delve into micro-optimising through assembly and intrinsic substitutes.

Having explored a range of serial optimisation options, we shift our focus to parallelism. Elfin was ported to OpenMP4[63] to take advantage of multithreaded processing on both CPUs and GPUs. The port to target GPUs faced a few hurdles, because C++ classes are not well supported by OpenMP. In the parallel loop region, any template-based containers (e.g. std::vector) must have their data pointer extracted as a primitive pointer. After mapping these data to the target device, the memory region may not resize. A further problem was with the calling of any class methods. OpenMP required explicit function pointer declaration for each class method called inside the parallel region, which is a problem that we found not well documented. Despite these slight setbacks, we were able to execute Elfin on several Nvidia GPUs.

Multi-processing tremendously sped up Elfin because the GA is embarrassingly parallel, meaning it scales well with the increase in number of processors. Elfin is clearly compute bound – each individual of the GA only holds N integers for a length-N module sequence, each identifying a specific protein module at a specific position, and a float point for its fitness score. The rest of the data have been either kept in a very compact relationship lookup table, or computed on the fly (such as CoMs that almost always change in each generation). For each byte of data in the GA individual there are many more computations done than copying and creating the data. The mutagenesis operators are perhaps the best example, where mutations can take several tries just to produce a viable offspring.

Lastly, OpenCL and MPI ports were considered. An MPI port would enable multi-processing across a cluster, and is likely an inevitable path for Elfin. In contrast, OpenCL requires kernelisation, the optimisation of which can make future changes to the algorithm a problematic task. Since we are expecting changes and maybe even entirely different meta-heuristics, OpenCL did not fit well at this stage of the development.

4. Results

4.1. Design Model Accuracy

Best and worst case benchmark results of each protein category are presented in Table 1. The full result table for all 30 designs can be found in Appendix F. In this section we first discuss the implications of these results on Elfin and its core algorithms. After that, we analyse the relaxed structures against Elfin's design outputs to shed light on whether the Protein LEGO design model is feasible.

In our experiment, Elfin was able to correctly solve all 20 randomly generated tests (i.e. to reach design scores of zero) after just one run per input. During these runs, the GA was set to stop if a score of zero has been found, or if 50 generations went by without any score improvement. For the RL-30 sequences, Elfin searched no more than a hundred generations with 524288 sequences in the population. Out of the 4.630 possible combinations in RL-30 sequences, only a total of less than 109 had to be sampled before Elfin converged. This is evidence of successful implementation of the GA and effective design of its mutagenesis operators.

On the other hand, while the Kabsch RMSD design scores can objectively quantify one solution's fitness, it does not tell whether a better solution exists (scores are available in Appendix F). It is therefore difficult to judge whether the solutions Elfin found for the hand-drawn cases are the best possible sequences. By visually inspecting the HD category of designs in Table 1 and

Table 2 however, we can at least be confident that Elfin has produced designs that are consistently congruent to their specifications across the variety of input problems we tested. This is despite the fact that some difficult cases were deliberately introduced (see 'B', 'R', 'T', 'spiral', and 'infinity'). These shapes are difficult because parts are present where nearby protein modules may be too close to be considered collision-free (or even interaction free for that matter). For instance, 'T' was the test of extreme bending (180o) and 'spiral' tested the ability to design a half-knot where the central tip of the protein is faced with possible collisions from all surrounding nearby modules. In the 'R' case, Elfin's design exhibits a clear bump just before its two 'legs' converge. This might be an artefact of the collision-phobic construction algorithm, which is a potential area Elfin can improve in (e.g. by allowing penalised collisions). In spite of these difficulties, Elfin was not prevented from delivering visually congruent designs. Thus, we consider the designs for hand-drawn cases adequate for further verification.

The majority of the relaxed protein models are extremely close to their designed counterparts. This is especially true for the random sequence benchmarks. The best benchmark model has a worst local

distortion of just 1.4Å (RL-10 2vnd, Table 1) at a length of 2082 residues, and at 4994 residues the RL-30 vi31 only deviates by a worst local distortion of 4.7Å. These values mean that even though the structures are extremely large, none of the local segments within these proteins move significantly away from their expected positions after rigorous sidechain repack and energy minimisation. Although, this is not to say that there are no lever effects or breakage present in the results. By comparing each designed model against their relaxed structure in 3D using PyMOL , it was clear that several designs have placed too much strain on certain parts of the protein such that loops were displaced. One such example is the 'horns' shape in

Table 2: on one side (left), the first horn is properly in place whereas on the other side (right), the horn was bent inwards during relaxation. Another similar case is 'R' from Table 1, in which the right 'leg' of 'R' is bent towards the left instead. In the second case however, it might not be solely due to a weak loop. Since the false bend takes place near the convergence of the two 'legs' of 'R', physical forces from nearby modules could have contributed to this structural failure. There are much to be hypothesised about just by visualising the differences of relaxed and designed protein structures, The important message here is that a protein engineer will be able to use Elfin to design very large proteins, and be able to study which protein module tend to fail and under what conditions (e.g. lever length, presence of nearby modules, and neighbour identities). These new structural knowledge will help advance current understanding of DHRs, but might only manifest when they are part of a larger design.

Nonetheless, our results contribute to positive progress in the direction of building very large proteins. In the best cases, our designed single-chain proteins are many times larger than any previously designed single-chain proteins. One would not expect a simple concatenation of multiple proteins to fold predictably, but that is not too different from what we have achieved using prior knowledge about repeat proteins. This result substantiates that the Protein LEGO design model is a feasible one and that our assumptions about repeat protein properties were not misled.

Table 1. Elfin's best and worst designed benchmark results of each category. RL-N stands for 'random length-N'. HD stands for 'hand drawn'. Design score per module for HD (L) is 19 and 102 for HD (R). All RL-N benchmarks scored 0.

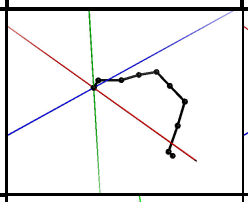
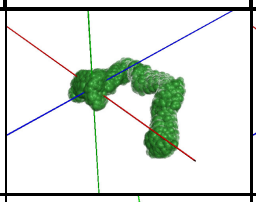
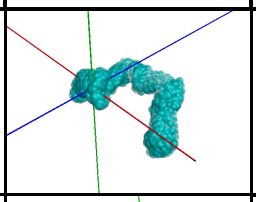
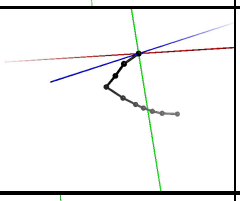
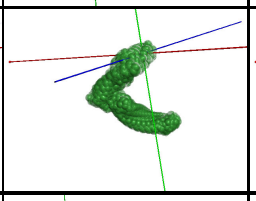
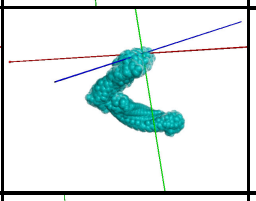
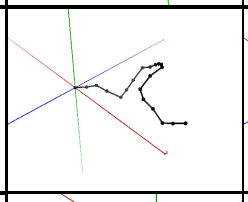
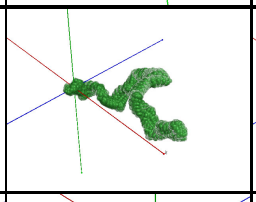
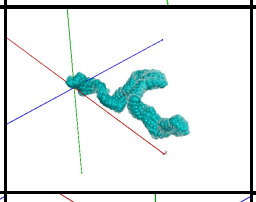
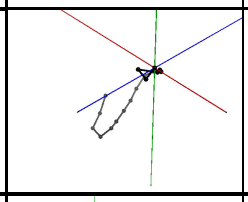
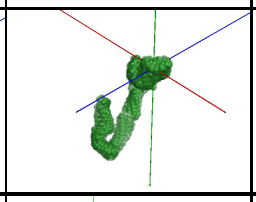
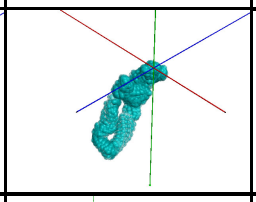
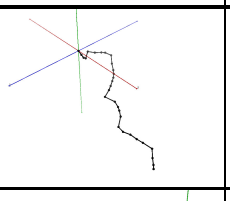
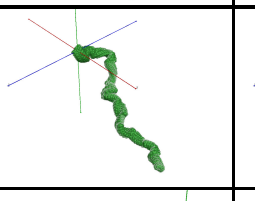
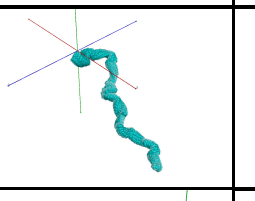
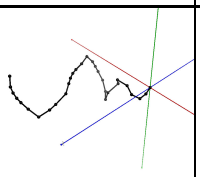
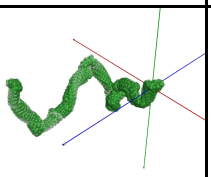
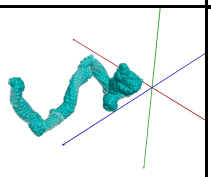
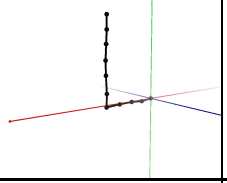
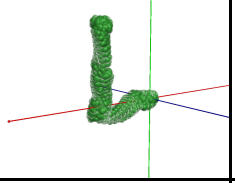
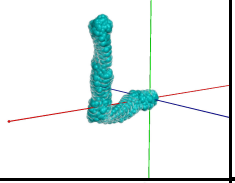
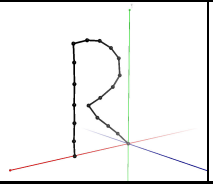
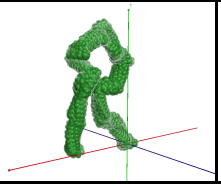
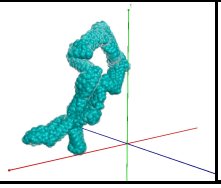
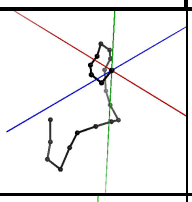
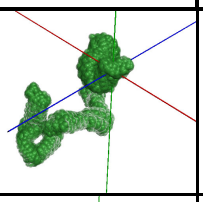
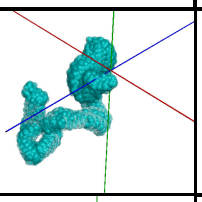
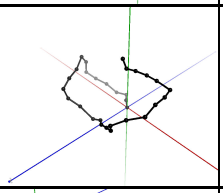
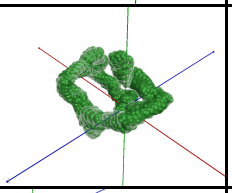
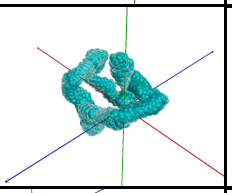
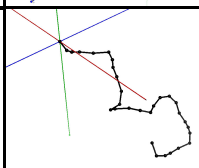
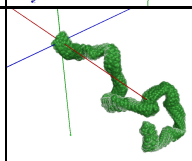
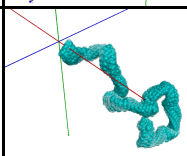
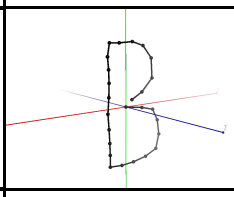
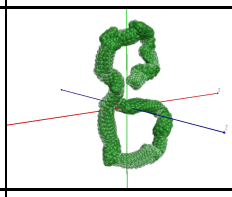
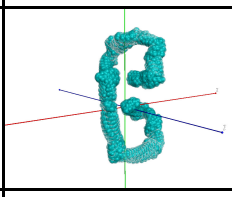
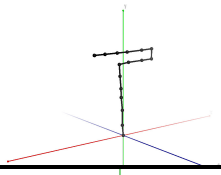
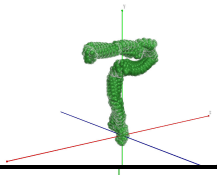
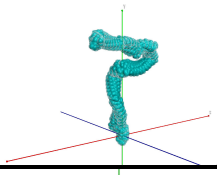
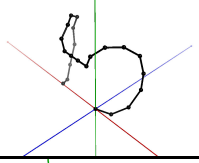
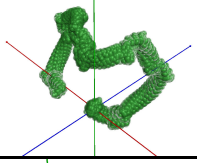
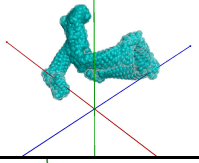
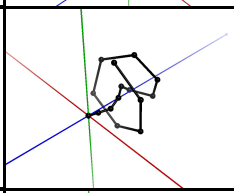
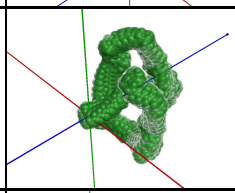
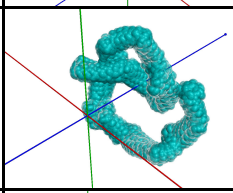
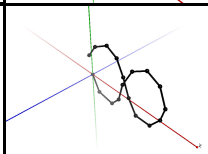
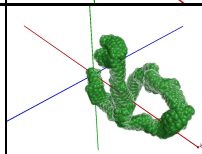
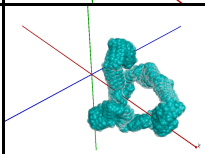
Name	Specification	Design	Relax	Max. Win. RMSD (Å)	No. of Residues
Best RL-10 (2vnd)				1.4	2082
Worst RL-10 (hfw4)				2.9	1504
Best RL-20 (1z1q)				3.9	3635
Worst RL-20 (cref)				7.0	3221
Best RL-30 (vias)				3.7	4641
Worst RL-30 (7r78)				6.6	4788
Best HD (L)				1.7	1661
Worst HD (R)				11.6	4003

***Table 2.Elfin's less representative design benchmark results. These are used to support texts referring to designs other than the ones in** Table 1.*

Name	Specification	Design	Relax	Max. Win. RMSD (Å)	No. of Residues
RL-20 (kwkh)				4.8	3870
RL-30 (xsx0)				4.9	5480
RL-30 (vi31)				4.7	4994
HD (B)				9.8	5162
HD (T)				5.0	2713
HD (Horns)				10.3	4181
HD (Spiral)				8.6	3767
HD (Infinity)				10.8	4872

4.2. Performance

A runtime measurement of Elfin was conducted across several different CPUs and GPUs. Since Elfin makes heavy use of random number generators and floating point operations, executing on different platforms or being compiled by different compilers could introduce a difference in the total number of iterations required for solving a particular design problem. Therefore, during our experiments Elfin was configured to always process 50 generations, with a population of 524288, and use the 'B' letter shape input. This number of generations ensured a very low score solution while keeping the amount of computation roughly equal. For each CPU benchmark, we compiled Elfin with GCC 6.1.0 without architecture-specific optimisation flags. GPU ports were compiled using the CLANG-YKT[12] compiler. The result execution times are plotted in *Figure 7*.

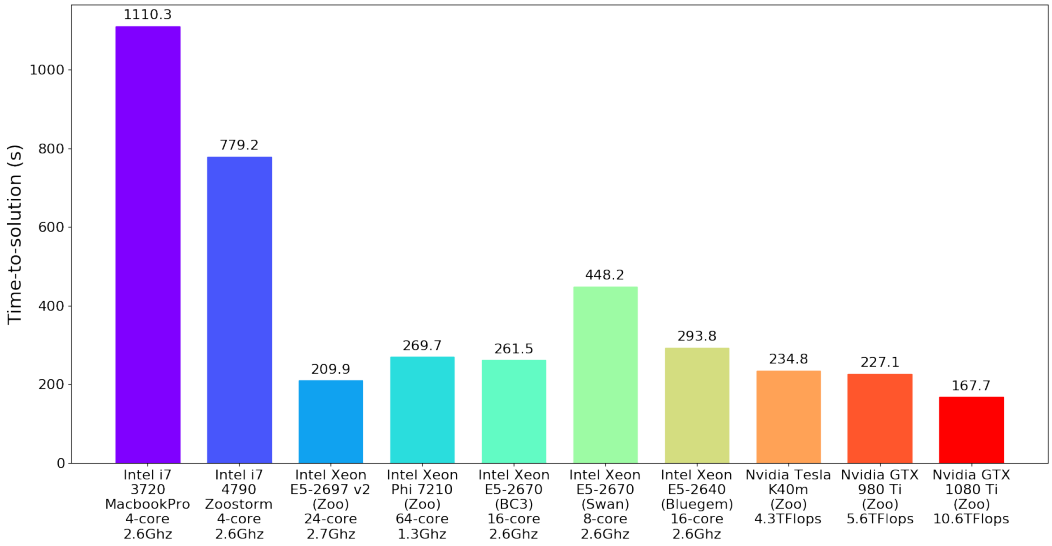


Figure 7. Benchmark execution times for 50 iterations on the 'B' design problem. Population size was set at 524288. Smaller numbers are better. Names in brackets are the compute clusters. BC3 stands for Bluecrystal Phase 3.

In the figure, MacbookPro (macOS Sierra,) and Zoostorm (Windows 10) reflect Elfin's performance on consumer grade laptop CPUs. The Intel Xeons on three different clusters represent typical server CPUs (different flavours of Linux). Lastly, GPUs are covered by the 2013 compute card Tesla K40m and two high-end consumer discrete GPU cards GTX 980 Ti (2015) and GTX 1080 Ti (2017).

[12] https://github.com/clang-ykt

It can be seen in *Figure 7* that the GPUs outperformed most of the CPUs, except the 24-core Xeon E5-2798 v7 node from the Zoo. This was not surprising because as already mentioned earlier, Elfin is a compute intensive, easily parallelised code. However, based purely on GFLOPS comparisons Elfin does not seem to be scaling very well. Part of the problem with Elfin being run on GPUs is the large number of divergent branches and non-deterministic code paths. These are especially detrimental for codes running on GPUs because each warp would execute all branches serially, leading considerable overheads and inefficient scheduling. It should however be noted that no special optimisation was performed for the GPU port. The same Elfin code was able run on all the different operating systems and on both CPUs and GPUs. Even without optimisation, the consumer grade GPUs are shoulder to shoulder with the best server CPUs we tested. This result inspires future work on Elfin to optimised for GPUs so that Elfin can be more accessible on personal computers. The first order of tasks in optimising Elfin for GPUs would be to reduce branch divergence by converting stochastic mutagenesis to use hard population ratio cut-offs. By doing so, the difference in computation between nearby individuals (in memory) could be minimised.

At its current state, Elfin took over 18 minutes to process 50 iterations on a mid-2012 MacbookPro, and just under 13 minutes on the Zoostorm (2015). These times would have been much worse had Elfin not take advantage of OpenMP. The speed at which Elfin is able to produce useful design results is not yet near interactive speeds, but Elfin is only the first version and improvements will certainly follow despite the closure of this project. It should be stressed also that these benchmark runs are using rather large population sizes, and each run on the different machines has reached their minimum sequence score well before the benchmark stopped timing. The results allow us to appreciate how long Elfin takes for a fixed amount of work on each machine. If anything, these numbers represent more difficult design cases rather than the typical design times. Furthermore, the user's configuration for Elfin ultimately dictates how long it takes to finish.

5. Conclusion

We have presented Elfin, an application developed for designing very large protein structures with practical efficiency. Through analysis of the 30 designs produced using Elfin, we were able to confirm our hypothesis on repeat protein units being well-behaved, rigid building blocks. This has in turn demonstrated the feasibility of the proposed Protein LEGO design model, as well as the effectiveness of the module sequence search algorithm we have developed. Our positive results encourage further investigations into the use of repeat proteins as building blocks in nanostructures. However, these massive proteins still await experimental verification to tell whether they are synthesisable. In addition and somewhat unexpectedly, we learned that building large protein structures from repeat units can help identify module properties otherwise not observable in isolated conditions. This is for instance measuring lever strain limits on a particular module, or finding correlation between repeat unit structure and module weakness, which would be of interest to repeat protein designers.

We hope that our work will inspire research in algorithm improvements and in identifying more useful properties of repeat proteins. In fact, Elfin's building strategy is not limited to repeat protein building blocks, as the assumptions used are identical to general rigid object pairs. Moreover, the shape specification used by Elfin can be extended to more complex, tree-like shapes by using tree data structures. Additional scoring functions can reward correct or incorrect positioning of specific modules that are related to activity sites.

Lastly, we gave an account of Elfin's optimisations and our effort in minimising the time-to-solution using parallel computing. It now runs at speeds suitable for practical research use on a single personal computer, but is a distance way from providing interactive design experience. Nevertheless, we identified room for improvement in its performance. Next up in this line of work will be an MPI port and incorporating Elfin into the Rosetta Software Suite. In conclusion, we believe Elfin was a successful proof of concept for the Protein LEGO design model, and exciting potentials lie ahead in using Elfin to gain more insight into the future of protein design.

References

[1] B. Kobe and A. V Kajava, “When protein folding is simplified to protein coiling: the continuum of solenoid protein structures,” *Trends Biochem. Sci.*, vol. 25, no. 10, pp. 509–515, 2000.

[2] E. M. Marcotte, M. Pellegrini, T. O. Yeates, and D. Eisenberg, “A census of protein repeats,” *J. Mol. Biol.*, vol. 293, no. 1, pp. 151–160, 1999.

[3] K. Park, B. W. Shen, F. Parmeggiani, P.-S. Huang, B. L. Stoddard, and D. Baker, “Control of repeat-protein curvature by computational protein design,” *Nat. Struct. Mol. Biol.*, vol. 22, no. 2, pp. 167–174, 2015.

[4] F. Parmeggiani and P.-S. Huang, “Designing repeat proteins: a modular approach to protein design,” *Curr. Opin. Struct. Biol.*, vol. 45, pp. 116–123, 2017.

[5] T. Aksel, A. Majumdar, and D. Barrick, “The contribution of entropy, enthalpy, and hydrophobic desolvation to cooperativity in repeat-protein folding,” *Structure*, vol. 19, no. 3, pp. 349–360, 2011.

[6] L. K. Mosavi, D. L. Minor, and Z. Peng, “Consensus-derived structural determinants of the ankyrin repeat motif,” *Proc. Natl. Acad. Sci.*, vol. 99, no. 25, pp. 16029–16034, 2002.

[7] Y. Javadi and L. S. Itzhaki, “Tandem-repeat proteins: regularity plus modularity equals design-ability,” *Curr. Opin. Struct. Biol.*, vol. 23, no. 4, pp. 622–631, 2013.

[8] J. Goyette and K. Gaus, “Mechanisms of protein nanoscale clustering,” *Curr. Opin. Cell Biol.*, vol. 44, pp. 86–92, 2017.

[9] I. Bethani, S. S. Skånland, I. Dikic, and A. Acker-Palmer, “Spatial organization of transmembrane receptor signalling,” *EMBO J.*, vol. 29, no. 16, pp. 2677–2688, 2010.

[10] J. B. Casaletto and A. I. McClatchey, “Spatial regulation of receptor tyrosine kinases in development and cancer,” *Nat. Rev. Cancer*, vol. 12, no. 6, pp. 387–400, 2012.

[11] A. Shaw *et al.*, “Spatial control of membrane receptor function using ligand nanocalipers,” *Nat. Methods*, vol. 11, no. 8, pp. 841–846, 2014.

[12] M. J. Dalby, N. Gadegaard, and R. O. C. Oreffo, “Harnessing nanotopography and integrin-matrix interactions to influence stem cell fate,” *Nat. Mater.*, vol. 13, no. 6, pp. 558–569, 2014.

[13] M. D. Tyka *et al.*, “Alternate states of proteins revealed by detailed energy landscape mapping,” *J. Mol. Biol.*, vol. 405, no. 2, pp. 607–618, 2011.

[14] B. Kuhlman, G. Dantas, G. C. Ireton, G. Varani, B. L. Stoddard, and D. Baker, “Design of a novel globular protein fold with atomic-level accuracy,” *Science (80-.).*, vol. 302, no. 5649, pp. 1364–1368, 2003.

[15] T. M. Jacobs *et al.*, “Design of structurally distinct proteins using strategies inspired by evolution,” *Science (80-.).*, vol. 352, no. 6286, pp. 687–690, 2016.

[16] P. B. Harbury, J. J. Plecs, B. Tidor, T. Alber, and P. S. Kim, “High-resolution protein design with backbone freedom,” *Science (80-.).*, vol. 282, no. 5393, pp. 1462–1467, 1998.

[17] P. B. Harbury, B. Tidor, and P. S. Kim, “Repacking protein cores with backbone freedom: structure prediction for coiled coils,” *Proc. Natl. Acad. Sci.*, vol. 92, no. 18, pp. 8408–8412, 1995.

[18] A. G. Murzin, A. M. Lesk, and C. Chothia, “Principles determining the structure of $β$-sheet barrels in

proteins II. The observed structures," *J. Mol. Biol.*, vol. 236, no. 5, pp. 1382–1400, 1994.

[19] C. W. Wood *et al.*, "CCBuilder: an interactive web-based tool for building, designing and assessing coiled-coil protein assemblies," *Bioinformatics*, p. btu502, 2014.

[20] B. Gutte, "A synthetic 70-amino acid residue analog of ribonuclease S-protein with enzymic activity.," *J. Biol. Chem.*, vol. 250, no. 3, pp. 889–904, 1975.

[21] J. W. Ponder and F. M. Richards, "Tertiary templates for proteins: use of packing criteria in the enumeration of allowed sequences for different structural classes," *J. Mol. Biol.*, vol. 193, no. 4, pp. 775–791, 1987.

[22] J. R. Desjarlais and T. M. Handel, "De novo design of the hydrophobic cores of proteins," *Protein Sci.*, vol. 4, no. 10, pp. 2006–2018, 1995.

[23] B. I. Dahiyat and S. L. Mayo, "De Novo Protein Design: Fully Automated Sequence Selection," *Science (80-.).*, vol. 278, no. 5335, pp. 82–87, 1997.

[24] J. R. Desjarlais and N. D. Clarke, "Computer search algorithms in protein modification and design," *Curr. Opin. Struct. Biol.*, vol. 8, no. 4, pp. 471–475, 1998.

[25] N. Koga *et al.*, "Principles for designing ideal protein structures," *Nature*, vol. 491, no. 7423, pp. 222–227, 2012.

[26] Y.-W. Lin, S. Nagao, M. Zhang, Y. Shomura, Y. Higuchi, and S. Hirota, "Rational design of heterodimeric protein using domain swapping for myoglobin," *Angew. Chemie Int. Ed.*, vol. 54, no. 2, pp. 511–515, 2015.

[27] E. Marcos *et al.*, "Principles for designing proteins with cavities formed by curved $β$ sheets," *Science (80-.).*, vol. 355, no. 6321, pp. 201–206, 2017.

[28] F. H. C. Crick, "The Fourier transform of a coiled-coil," *Acta Crystallogr.*, vol. 6, no. 8–9, pp. 685–689, 1953.

[29] G. Grigoryan and W. F. DeGrado, "Probing designability via a generalized model of helical bundle geometry," *J. Mol. Biol.*, vol. 405, no. 4, pp. 1079–1100, 2011.

[30] A. R. Thomson *et al.*, "Computational design of water-soluble $α$-helical barrels," *Science (80-.).*, vol. 346, no. 6208, pp. 485–488, 2014.

[31] G. Bhardwaj *et al.*, "Accurate de novo design of hyperstable constrained peptides," *Nature*, 2016.

[32] E. a Coutsias, C. Seok, M. P. Jacobson, and K. a Dill, "A kinematic view of loop closure," *J. Comput. Chem.*, vol. 25, no. 4, pp. 510–528, 2004.

[33] P. Gainza, H. M. Nisonoff, and B. R. Donald, "Algorithms for protein design," *Curr. Opin. Struct. Biol.*, vol. 39, pp. 16–26, 2016.

[34] C. H. Norn and I. André, "Computational design of protein self-assembly," *Curr. Opin. Struct. Biol.*, vol. 39, pp. 39–45, 2016.

[35] R. Veneziano *et al.*, "Designer nanoscale DNA assemblies programmed from the top down," *Science (80-.).*, vol. 352, no. 6293, p. 1534, 2016.

[36] S. M. Douglas, A. H. Marblestone, S. Teerapittayanon, A. Vazquez, G. M. Church, and W. M. Shih, "Rapid prototyping of 3D DNA-origami shapes with caDNAno," *Nucleic Acids Res.*, p. gkp436, 2009.

[37] C. D. Schenkelberg and C. Bystroff, "InteractiveROSETTA: a graphical user interface for the PyRosetta

protein modeling suite," *Bioinformatics*, p. btv492, 2015.

[38] A. V Kajava, "Tandem repeats in proteins: from sequence to structure," *J. Struct. Biol.*, vol. 179, no. 3, pp. 279–288, 2012.

[39] X. Wang, J. McLachlan, P. D. Zamore, and T. M. T. Hall, "Modular recognition of RNA by a human pumilio-homology domain," *Cell*, vol. 110, no. 4, pp. 501–512, 2002.

[40] A. N.-S. Mak, P. Bradley, R. A. Cernadas, A. J. Bogdanove, and B. L. Stoddard, "The crystal structure of TAL effector PthXo1 bound to its DNA target," *Science (80-.).*, vol. 335, no. 6069, pp. 716–719, 2012.

[41] D. Deng *et al.*, "Structural basis for sequence-specific recognition of DNA by TAL effectors," *Science (80-.).*, vol. 335, no. 6069, pp. 720–723, 2012.

[42] A. Barkan *et al.*, "A combinatorial amino acid code for RNA recognition by pentatricopeptide repeat proteins," *PLoS Genet*, vol. 8, no. 8, p. e1002910, 2012.

[43] C. Reichen, S. Hansen, and A. Plückthun, "Modular peptide binding: from a comparison of natural binders to designed armadillo repeat proteins," *J. Struct. Biol.*, vol. 185, no. 2, pp. 147–162, 2014.

[44] H. K. Binz *et al.*, "High-affinity binders selected from designed ankyrin repeat protein libraries," *Nat. Biotechnol.*, vol. 22, no. 5, pp. 575–582, 2004.

[45] G. Varadamsetty, D. Tremmel, S. Hansen, F. Parmeggiani, and A. Plückthun, "Designed Armadillo repeat proteins: library generation, characterization and selection of peptide binders with high specificity," *J. Mol. Biol.*, vol. 424, no. 1, pp. 68–87, 2012.

[46] A. L. Cortajarena, T. Y. Liu, M. Hochstrasser, and L. Regan, "Designed proteins to modulate cellular networks," *ACS Chem. Biol.*, vol. 5, no. 6, p. 545, 2010.

[47] T. J. Brunette *et al.*, "Exploring the repeat protein universe through computational protein design," *Nature*, vol. 528, no. 7583, pp. 580–584, 2015.

[48] L. Doyle *et al.*, "Rational design of $α$-helical tandem repeat proteins with closed architectures," *Nature*, vol. 528, no. 7583, pp. 585–588, 2015.

[49] P.-S. Huang, K. Feldmeier, F. Parmeggiani, D. A. F. Velasco, B. Höcker, and D. Baker, "De novo design of a four-fold symmetric TIM-barrel protein with atomic-level accuracy," *Nat. Chem. Biol.*, vol. 12, no. 1, pp. 29–34, 2016.

[50] F. Parmeggiani *et al.*, "A general computational approach for repeat protein design.," *J. Mol. Biol.*, vol. 427, no. 2, pp. 563–575, 2015.

[51] H. Berman, K. Henrick, and H. Nakamura, "Announcing the worldwide protein data bank," *Nat. Struct. Mol. Biol.*, vol. 10, no. 12, p. 980, 2003.

[52] A. Leaver-Fay *et al.*, "ROSETTA3: an object-oriented software suite for the simulation and design of macromolecules," *Methods Enzymol.*, vol. 487, p. 545, 2011.

[53] P. H. Schönemann, "A generalized solution of the orthogonal Procrustes problem," *Psychometrika*, vol. 31, no. 1, pp. 1–10, 1966.

[54] W. Kabsch, "A solution for the best rotation to relate two sets of vectors," *Acta Crystallogr. Sect. A Cryst. Physics, Diffraction, Theor. Gen. Crystallogr.*, vol. 32, no. 5, pp. 922–923, 1976.

[55] D. T. Jones, "De novo protein design using pairwise potentials and a genetic algorithm," *Protein Sci.*,

vol. 3, no. 4, pp. 567–574, 1994.

[56] R. Unger and J. Moult, “Genetic algorithms for protein folding simulations,” *J. Mol. Biol.*, vol. 231, no. 1, pp. 75–81, 1993.

[57] G. A. Lazar, J. R. Desjarlais, and T. M. Handel, “De novo design of the hydrophobic core of ubiquitin,” *Protein Sci.*, vol. 6, no. 6, pp. 1167–1178, 1997.

[58] J. T. Pedersen and J. Moult, “Genetic algorithms for protein structure prediction,” *Curr. Opin. Struct. Biol.*, vol. 6, no. 2, pp. 227–231, 1996.

[59] J. Desmet, M. De Maeyer, B. Hazes, and I. Lasters, “The dead-end elimination theorem and its use in protein side-chain positioning.,” *Nature*, vol. 356, no. 6369, pp. 539–542, 1992.

[60] D. B. Gordon and S. L. Mayo, “Radical performance enhancements for combinatorial optimization algorithms based on the dead-end elimination theorem,” *J. Comput. Chem.*, vol. 19, no. 13, pp. 1505–1514, 1998.

[61] L. L. Looger and H. W. Hellinga, “Generalized dead-end elimination algorithms make large-scale protein side-chain structure prediction tractable: implications for protein design and structural genomics,” *J. Mol. Biol.*, vol. 307, no. 1, pp. 429–445, 2001.

[62] A. Fraser, D. Burnell, and others, “Computer models in genetics.,” *Comput. Model. Genet.*, 1970.

[63] OpenMP Architecture Review Board, “OpenMP Application Program Interface Version 4.0.” 2013.

Appendices

A. Software Used

Article	Source	Use or Modification
PyMOL Edu	Schrödinger LLC	Protein visualisation.
Rosetta Software Suite	Rosetta Commons	Protein refinement and scoring.
BioPython (library)	Python https://github.com/nlohmann/json	Protein data file (PDBs) manipulation.
JSON (library)	C++ https://github.com/nlohmann/json	Configuration, input, and output file parsing.
Elfin	C++/Python/C/Matlab/Shell https://github.com/joy13975/elfin	Student written software.

B. Journal Paper Abstract

Elfin: an algorithm for computational design of custom three-dimensional structures from modular repeat protein building blocks

Chun-Ting Yeh, TJ Brunette, David Baker, Simon McIntosh-Smith, Fabio Parmeggiani

Abstract

Computational protein design methods have enabled the design of novel protein structures, but they are often still limited to small proteins and symmetric systems.

Here we describe Elfin – a genetic algorithm for the design of novel proteins with custom shapes using structural building blocks derived from experimentally verified repeat proteins. By combining building blocks with compatible interfaces, it is possible to rapidly build non-symmetric large structures (> 1000 amino acids) that match three-dimensional geometric descriptions provided by the user. A run time of 13 minutes on a laptop computer for a 5162 amino acid structure (letter 'B' benchmark) and a linear scaling with protein size, make Elfin accessible to users with limited computational resources.

With a fast and reliable design process, protein structures with controlled geometry will allow the systematic study of the effect of spatial arrangement of enzymes and signalling molecules, and provide new scaffolds for functional nanomaterials.

C. First Grid Search Parameter Ranges

Parameter	Values
Chromosome Length Deviation	0.05, 0.1, 0.2, 0.3, 0.5
Survival Rate	0.005, 0.01, 0.02, 0.05, 0.1
Crossover Rate	0.1, 0.3, 0.5, 0.7, 0.9
Point Mutate Rate	0.25, 0.5, 0.75
Limb Mutate Rate	0.25, 0.5, 0.75

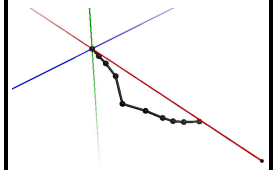
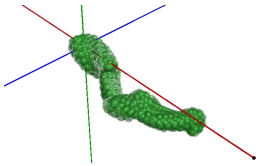
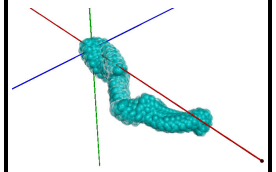
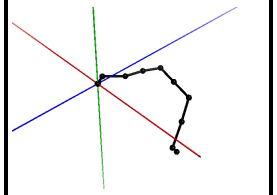
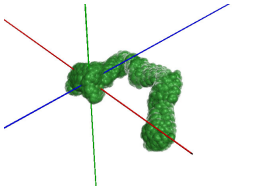
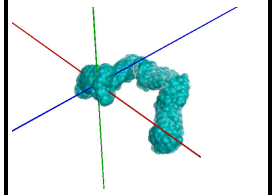
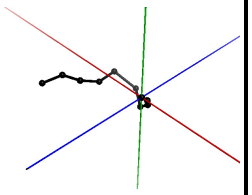
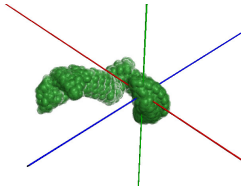
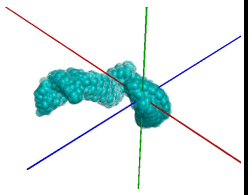
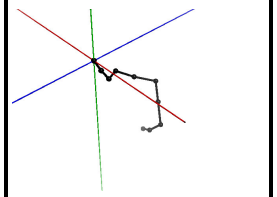
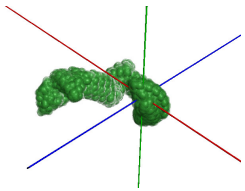
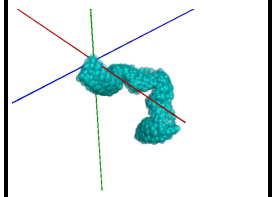
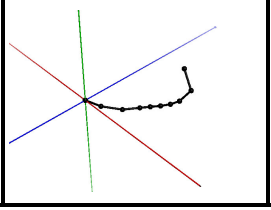
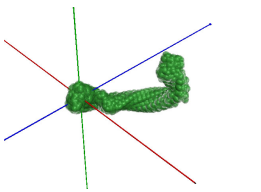
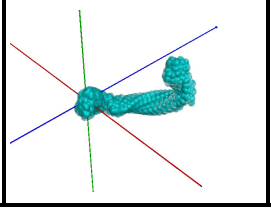
D. Second Grid Search Parameter Ranges

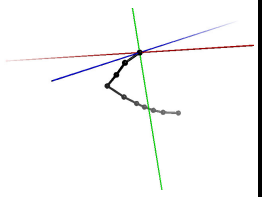
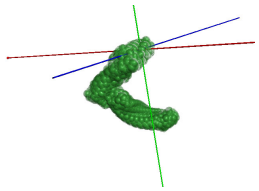
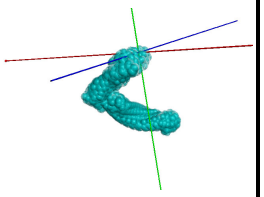
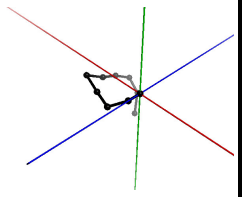
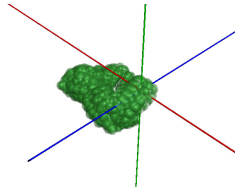
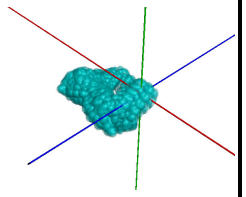
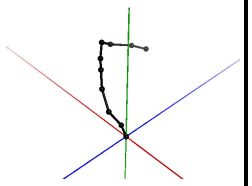
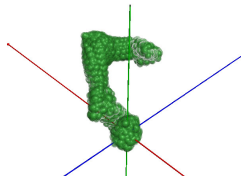
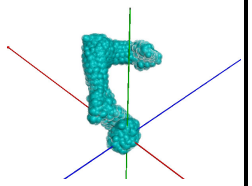
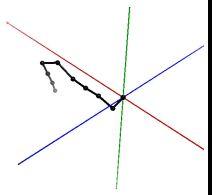
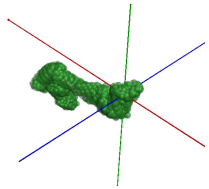
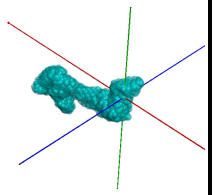
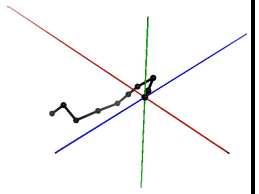
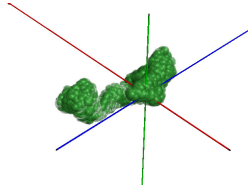
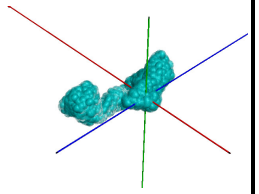
Parameter	Values
Chromosome Length Deviation	0.15, 0.2, 0.25
Survival Rate	0.02, 0.03, 0.05
Crossover Rate	0.2, 0.4, 0.6
Point Mutate Rate	0.3, 0.5, 0.7
Limb Mutate Rate	0.3, 0.5, 0.7

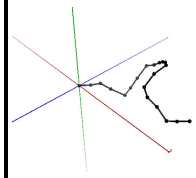
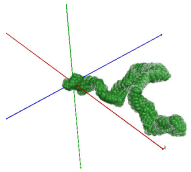
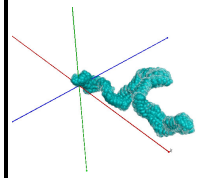
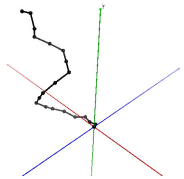
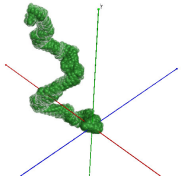
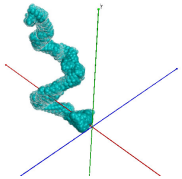
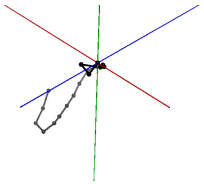
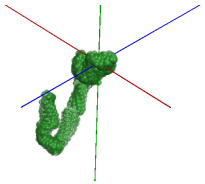
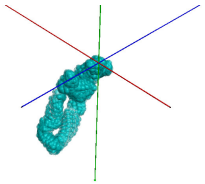
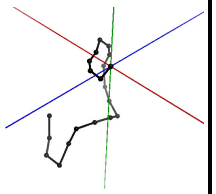
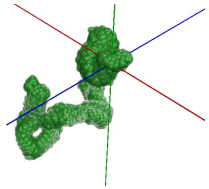
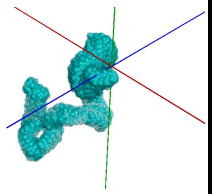
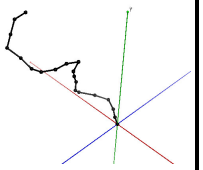
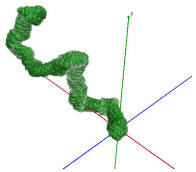
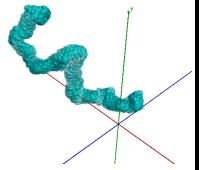
E. Optimal Elfin Parameters

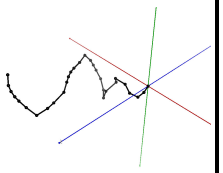
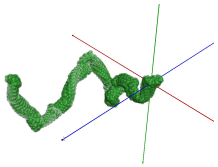
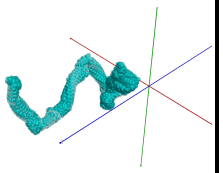
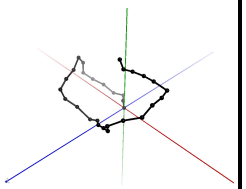
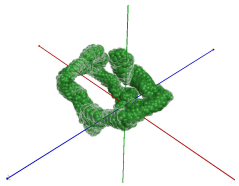
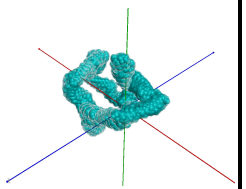
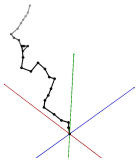
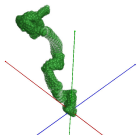
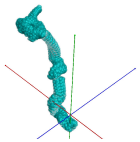
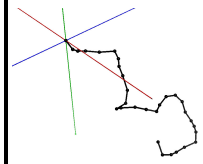
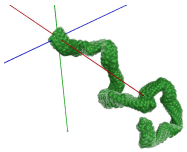
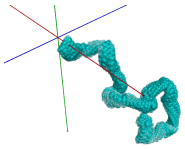
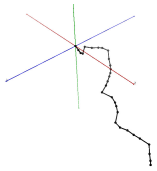
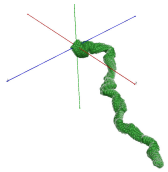
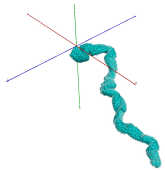
Parameter	Values
Chromosome Length Deviation	0.2
Survival Rate	0.02
Crossover Rate	0.2
Point Mutate Rate	0.5
Limb Mutate Rate	0.5

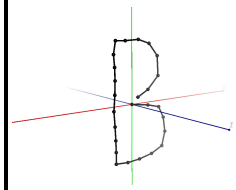
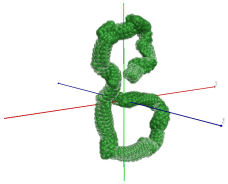
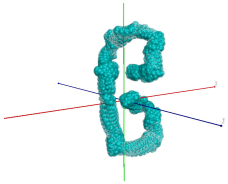
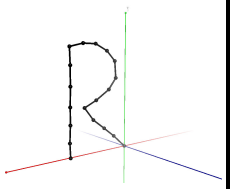
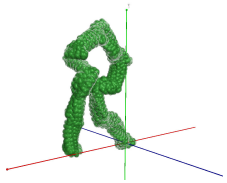
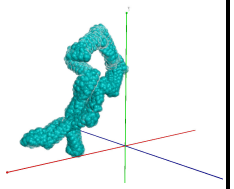
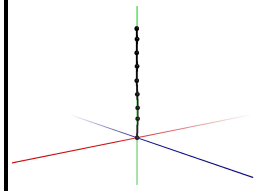
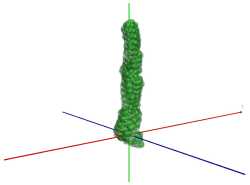
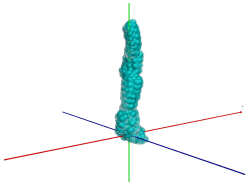
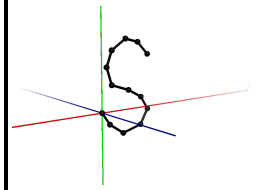
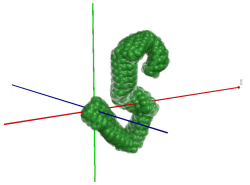
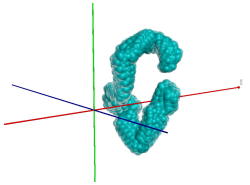
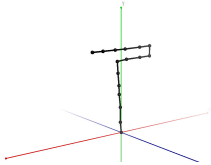
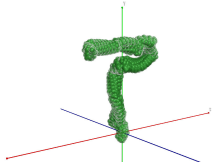
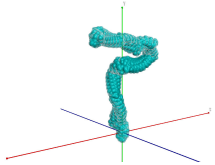
F. Full Benchmark Result Table

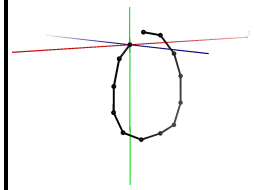
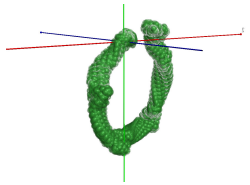
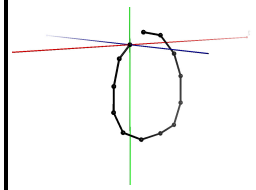
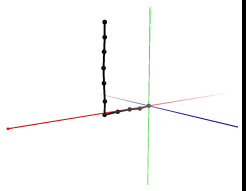
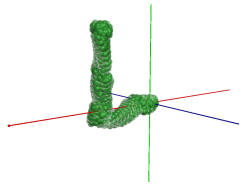
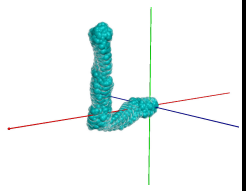
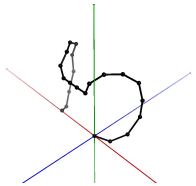
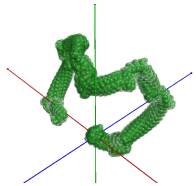
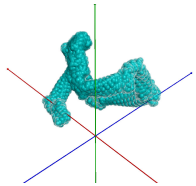
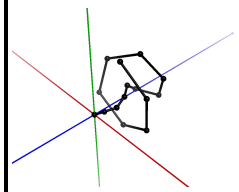
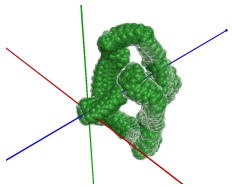
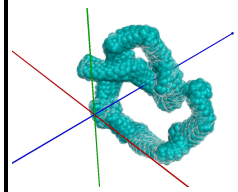
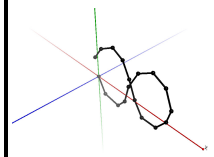
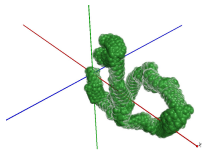
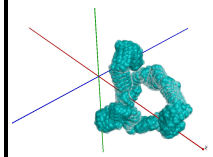
Name	Type	No. of Residues	Specification	Design	Relaxed	Design Score	Global RMSD (Å)	Max Windowed RMSD (Å)
2x7c	Random 10 Modules	1460				0	3.4	2.1
2vnd		2082				0	2.0	1.4
1696		1751				0	2.4	1.7
d0qe		1832				0	5.0	2.6
eabj		1532				0	1.8	1.7

Name	Type	No. of Residues	Specification	Design	Relaxed	Design Score	Global RMSD (Å)	Max Windowed RMSD (Å)
hfw4	Random 10 Modules	1504				0	5.9	2.9
iv57		1701				0	2.8	2.1
ju1z		1857				0	2.5	1.8
v722		1930				0	5.8	2.5
iz7z		1961				0	4.5	2.6

Name	Type	No. of Residues	Specification	Design	Relaxed	Design Score	Global RMSD (Å)	Max Windowed RMSD (Å)
1z1q	Random 20 Modules	3635				0	10.2	3.9
8gcv		3744				0	10.1	4.3
cref		3221				0	28.0	7.0
kwkh		3870				0	12.2	4.8
mqj2		3943				0	19.9	6.2

Name	Type	No. of Residues	Specification	Design	Relaxed	Design Score	Global RMSD (Å)	Max Windowed RMSD (Å)
7r78	Random 30 Modules	4788				0	23.3	6.6
xsx0		5480				0	13.9	4.9
ugyg		5322				0	23.8	6.5
vi31		4994				0	13.6	4.7
vias		4641				0	8.0	3.7

Name	Type	No. of Residues	Specification	Design	Relaxed	Design Score/Modules	Global RMSD (Å)	Max Windowed RMSD (Å)
B	Hand Drawn	5162				5602/32	51.0	9.8
R		4003				2457/24	70.3	11.6
I		1149				146/8	7.9	3.1
S		1971				509/14	39.3	7.4
T		2713				1697/19	11.5	5.0

Name	Type	No. of Residues	Specification	Design	Relaxed	Design Score/Modules	Global RMSD (Å)	Max Windowed RMSD (Å)
O	Hand Drawn	3175				1065/21	11.3	4.4
L		1661				227/12	2.4	1.7
Horns		4181				4108/27	53.2	10.3
Spiral		3767				1899/23	37.5	8.6
Infinity		4872				5126/27	52.2	10.8