IMPROVING COMPUTATIONAL METHODS FOR DESIGNING POLAR PROTEIN-PROTEIN INTERFACES

Jack Barton Maguire

A dissertation submitted to the faculty at the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Curriculum in Bioinformatics and Computational Biology.

Chapel Hill
2020

ABSTRACT

Jack Barton Maguire: Improving Computational Methods for Designing Polar Protein-Protein Interfaces
(Under the direction of Brian Kuhlman)

Computational protein design has come a long way over the past few decades, but there is still room to improve. Even state-of-the-art computational protein modeling software has challenges when attempting to design protein-protein interactions. Current rotamer optimization protocols have problems when performing sequence design at the interface of multiple protein chains, specifically with respect to desolvation penalties. Additionally, modern docking protocols use artificial energy landscapes that are poorly suited for protein interface design. In this study, I inspect the current state-of-the-art protocols, identify their shortcomings, and develop and benchmark improvements and/or replacements. I lay out three improvements (two for rotamer optimization at the interface and one for docking), benchmark them, and show that they all improve our ability to sample the energy landscape provided. The benchmarks show that, based on computational metrics, our new protocols are able to minimize risk of desolvation penalties without any energetic tradeoffs compared to existing standards.

TABLE OF CONTENTS

LIST OF FIGURES………………………………………………………………………….. ix

LIST OF TABLES …………………………………………………………………………... xi

LIST OF ABBREVIATIONS ………………………………………………………………… xii

CHAPTER 1: INTRODUCTION …………………………………………………………… 1

1.1 BACKGROUND ……………………………………………………………….. 1

1.2 OPPORTUNITIES FOR INNOVATION ……………………………………… 3

1.2.1 IMPROVED ROTAMER SAMPLING …………………………………… 3

1.2.2 IMPROVED BACKBONE SAMPLING EFFICIENCY ………………….. 4

1.3 CHAPTER ORGANIZATION AND READING ADVICE ………………………… 6

1.4 CONCESSIONS ……………………………………………………………….. 7

REFERENCES ……………………………………………………………………... 8

CHAPTER 2: RAPID SAMPLING OF HYDROGEN BOND NETWORKS FOR COMPUTATIONAL PROTEIN DESIGN ……………………………… 11

2.1 INTRODUCTION …………………………………………………………….. 11

2.2 METHODS ……………………………………………………………………… 13

2.2.1 SIDE CHAIN SAMPLING AND IDENTIFICATION OF HYDROGEN BONDS …………………………………………….. 13

2.2.2 HBONDGRAPH DATA STRUCTURE ………………………………… 15

2.2.3 MONTE CARLO HBONDGRAPH TRAVERSAL …………………… 16

2.2.4 RANKING RESULTS AND OUTPUT …………………………………. 19

2.2.4.1 SATURATION ………………………………………………… 19

2.2.4.2 HBNET SCORE ……………………………………………….. 20

2.2.4.3 OUTPUT ……………………………………………………. 20

2.2.4.4 BURIAL CALCULATIONS …………………………………… 21

2.2.5 BENCHMARKS AND ANALYSIS ……………………………………… 21

2.2.5.1 NATIVE NETWORK RECOVERY …………………………… 21

2.2.5.2 NETWORK DESIGN BENCHMARKS ………………………. 22

2.3 RESULTS AND DISCUSSION …………………………………………………… 24

2.3.1 BENEFITS OF EXTRA CHI SAMPLING ……………………………… 24

2.3.2 NEW DESIGN CASES ENABLED BY MC HBNET …………………… 25

2.3.3 SYMMETRIC HOMO-OLIGOMER BENCHMARKS ………………… 29

2.4 CONCLUSIONS ………………………………………………………………… 30

2.5 SUPPLEMENTAL INFORMATION ……………………………………………… 31

2.5.1 SUPPLEMENTARY METHODS ………………………………………... 31

2.5.1.1 ORIGINAL HBNET IMPLEMENTATION …………………… 31

2.5.1.2 MC HBNET RUNS ON BACKRUB TRAJECTORIES ……… 35

2.5.1.3 ROSETTA SCRIPT FOR THE THREE INTERFACE "MOTIVATING" CASES ……………………… 37

2.5.1.4 ROSETTA SCRIPT FOR THE TWO SYMMETRIC HOMODIMER CASES ………………………... 38

2.5.1.5 ROSETTA SCRIPT FOR THE TWO SYMMETRIC HOMOTRIMER CASES ……………………… 39

2.5.1.5 ROSETTA SCRIPT FOR THE SMALL HELICAL MONOMER "MOTIVATING" CASE ……………. 40

2.5.2 SUPPLEMENTARY FIGURES ………………………………………….. 41

2.5.3 SUPPLEMENTARY TABLES …………………………………………... 42

ENDNOTE …………………………………………………………………………... 46

REFERENCES ……………………………………………………………………… 47

CHAPTER 3: OPTIMIZING ENERGY LANDSCAPE PERTURBATIONS FOR IMPROVED PACKING DURING PROTEIN DESIGN ……………... 50

3.1 PREFACE ………………………………………………………………………... 50

3.2 BACKGROUND ……………………………………………………………… 50

3.3 DIAGNOSING THE PROBLEM ……………………………………………... 52

3.4 BENEFITS OF INCREASING REPULSIVE WEIGHT…………………………… 54

3.5 BENEFITS OF RAMPING REFERENCE WEIGHT……………………………… 56

3.6 METHODS ……………………………………………………………………… 58

3.6.1 PROTEIN STRUCTURE SETS ………………………………………….. 58

3.6.2 FASTDESIGN BENCHMARKS ………………………………………… 59

3.6.3 RELAXING CRYSTAL STRUCTURES ………………………………… 60

3.6.4 REFERENCE ENERGY FITTING ……………………………………… 60

3.7 CONCLUSION ………………………………………………………………….. 60

3.8 SUPPLEMENTAL INFORMATION ……………………………………………… 61

3.8.1 SCRIPT FOR SEWING DESIGNS ……………………………………… 61

3.8.2 COMMAND LINE FLAGS FOR SEWING DESIGNS ………………….. 61

REFERENCES ……………………………………………………………………… 62

CHAPTER 4: BENCHMARKING NEW COMPUTATIONAL TECHNIQUES FOR POLAR PROTEIN-PROTEIN INTERFACE DESIGN ……………… 64

4.1 INTRODUCTION ……………………………………………………………….. 64

4.2 AVERAGE TRAJECTORY TEST …………………………………………………… 65

4.2.1 METHODS …………………………………………………………………. 65

4.2.1.1 STRUCTURE GENERATION ………………………………… 65

4.2.1.2 RUNNING ROSETTA ………………………………………… 65

4.2.1.3 EVALUATING RESULTS ……………………………………. 66

4.2.2 RESULTS AND DISCUSSION ……………………………………………. 67

4.3 TOP TRAJECTORY TEST …………………………………………………………… 70

4.3.1 METHODS …………………………………………………………………. 70

4.3.1.1 STRUCTURE GENERATION ………………………………… 70

4.3.1.2 RUNNING ROSETTA ………………………………………… 70

4.3.1.3 EVALUATING RESULTS ……………………………………. 71

4.3.2 RESULTS AND DISCUSSION ……………………………………………. 71

4.4 CONCLUSION ……………………………………………………………………….. 73

4.5 SUPPLEMENTAL INFORMATION ………………………………………………… 74

4.5.1 ROSETTA SCRIPTS FOR "AVERAGE TRAJECTORY TEST" ………. 74

4.5.2 ROSETTA SCRIPTS FOR "TOP TRAJECTORY TEST" ………………. 83

REFERENCES ………………………………………………………………………………… 87

CHAPTER 5: USING A DEEP NEURAL NETWORK TO IMPROVE LOW-RESOLUTION MODELING OF PROTEIN-PROTEIN INTERACTIONS PRIOR TO DESIGN ……………………………………. 89

5.1 INTRODUCTION …………………………………………………………………….. 89

5.1.1 BACKGROUND …………………………………………………………… 89

5.1.2 PROJECT DESCRIPTION ………………………………………………… 90

5.2 METHODS …………………………………………………………………………… 91

5.2.1 RAY CASTING …………………………………………………………….. 91

5.2.1.1 ADDITIONAL DATA COLLECTED ………………………… 93

5.2.2 MOUSE IMPLEMENTATION ……………………………………….. 93

5.2.3 NEURAL NETWORK TRAINING …………………………………… 94

5.2.3.1 TRAINING DATA …………………………………………... 95

5.2.4 ONE-SIDED INTERFACE DESIGN BENCHMARK ………………... 95

5.3 RESULTS AND DISCUSSION ……………………………………………… 97

5.4 CONCLUSION AND FUTURE WORK …………………………………….. 98

5.5 SUPPLEMENTAL INFORMATION ………………………………………… 99

5.5.1 MOUSE GENERATION 1 NETWORK ARCHITECTURE …………….. 99

5.5.2 MOUSE GENERATION 2 NETWORK ARCHITECTURE …………… 100

5.5.3 VISUALIZATION OF WHAT MOUSE "SEES" ……………………… 101

REFERENCES ……………………………………………………………….. 103

LIST OF FIGURES

1.1 TIMELINE OF MAJOR ROSETTA DEVELOPMENTS ………………………………….. 3

1.2 TOY ILLUSTRATION OF OUR IDEA FOR IMPROVED
BACKBONE SAMPLING EFFICIENCY ……………………………………………… 5

2.1 HBONDGRAPH ……………………………………………………………………… 15

2.2 MONTE CARLO GROWTH OF A NETWORK …………………………………………... 18

2.3 NATIVE NETWORKS: IMPACT OF EXTRA CHI SAMPLING ………………………… 25

2.4 AGGREGATE DATA FROM ONE-SIDED INTERFACE DESIGN BENCHMARKS ….. 26

2.5 NEW DESIGN PROBLEMS ………………………………………………………….. 27

2.6 SYMMETRIC INTERFACES ………………………………………………………... 29

2.S1 SMALL BACKBONE CHANGES AFFECT POSSIBLE HYDROGEN
BONDS AND NETWORK CONNECTIVITIES ……………………………………… 41

3.1 FASTRELAX'S DEFAULT REPULSIVE RAMPING SCHEME ………………………… 51

3.2 AMINO ACID DISTRIBUTION OF NATIVE PROTEINS AND
PROTEINS DESIGNED BY FASTDESIGN …………………………………………... 52

3.3 DESCRIPTION OF FASTDESIGN'S BIAS AND THE MEANS TO ADDRESS IT …….. 54

3.4 RESULTS OF HYPERPARAMETER SAMPLING FOR
TWO OF OUR DESIGN CASES ……………………………………………………… 55

3.5 AMINO ACID DISTRIBUTIONS FOR VARIOUS BENCHMARKS
FOR NATIVE PROTEINS AND FOR DESIGNS OF THREE
DIFFERENT FASTDESIGN PROTOCOLS …………………………………………... 57

3.6 COMPARISON OF NEW DESIGN PROTOCOLS WITH LEGACY
FOR THREE DESIGN CASES ……………………………………………………… 58

5.1 VISUALIZATION OF RAY CASTING GEOMETRIES THAT ARE
FED INTO THE NEURAL NETWORK ……………………………………………… 92

5.2 CORRELATION DISTRIBUTIONS BETWEEN SEVERAL
LOW-RESOLUTION METRICS AND THE FINAL
HIGH-RESOLUTION SCORE AFTER FASTDESIGN …………………………… 97

5.S1 REALISTIC VISUALIZATION OF HOW MOUSE SEES DISTANCE ……………… 101

5.S2 HD VISUALIZATION OF HOW MOUSE SEES RELATIVE ORIENTATION ……… 102

LIST OF TABLES

2.1 DEFINITIONS OF EXTRA-CHI SAMPLING LEVELS …………………………………. 14

2.S1 DATA FOR “SMALL INTERFACE” CASE …………………………………………... 42

2.S2 DATA FOR “MEDIUM INTERFACE” CASE ………………………………………… 42

2.S3 DATA FOR “LARGE INTERFACE” CASE …………………………………………... 43

2.S4 DATA FOR “SMALL HELICAL MONOMER” CASE ………………………………… 43

2.S5 DATA FOR “SYMMETRIC HOMODIMER 5J0K” CASE …………………………… 44

2.S6 DATA FOR “SYMMETRIC HOMODIMER 5J10” CASE …………………………… 44

2.S7 DATA FOR “SYMMETRIC HOMODIMER 5J0H” CASE …………………………… 45

2.S8 DATA FOR “SYMMETRIC HOMODIMER 5IZS” CASE …………………………… 45

3.1 HYPERPARAMETERS AND REPULSIVE WEIGHTS FOR OUR
FOUR FINAL PROTOCOLS ……………………………………………………... 56

3.2 RESULTS OF RUNNING NEW PROTOCOLS ON OUR
DIVERSE COLLECTION OF DESIGN CASES …………………………………… 56

4.1 COMPUTATIONAL DESIGN QUALITY METRICS FOR
NATIVE INTERFACE REDESIGNS USING DIFFERENT METHODS …………….. 68

4.2 RESULTS OF THE “TOP TRAJECTORY TEST” ……………………………………… 72

LIST OF ABBREVIATIONS

MOUSE	Model Of Ultimate Surface Energy
REU	Rosetta Energy Units
RG	Radius of Gyration
SEWING	Structure Extension With Native fragment-Graphs

CHAPTER 1: Introduction

1.1 Background

The ability to design a protein-protein interface gives scientists the ability to engineer interactions within living organisms. Scientists can design novel proteins that bind to native proteins in efforts to control chemical reactions, target diseases, and provide us with a better understanding of biological systems.[1] Biochemists have historically engineered protein-protein interactions using a mixture of rational design, homologous sequence alignments, and high-throughput experimental testing.[2,3] These techniques are considered relatively conservative because they only introduce a handful of mutations to native interfaces; they are impractical for a large-scale redesign of a native interface or the design of a de novo interface. Over the past few decades, scientists have had increasing success using computers to design protein interfaces in a less conservative manner.[4–10]

The Rosetta protein-modeling suite[11] has been used to design protein-protein interfaces in both one-sided (where only one of the two protein chains is able to be mutated) and two-sided cases.[12–17] The general interface design protocol in Rosetta involves rigid-body docking followed by alternating phases of fixed-backbone sidechain sampling and all-atom energy minimization.[18,19] Rosetta introduces mutations during the sidechain sampling phase and are subject to restrictions set by the user.

One major challenge associated with protein-protein interface design is the handling of polar atoms that are buried by the interface.[8,12,20] To offset the penalty in desolvation energy of binding, all polar atoms at the interface must have a hydrogen bonding partner in the bound state. The penalty for a single unsatisfied polar atom ("unsat" for short, meaning a polar atom with no hydrogen bonding partner) is estimated to be 3 kcal/mol.[21] For this reason, any computational strategy for designing protein-protein interfaces should aim to minimize the number of buried unsats. It is rarely an acceptable solution to simply restrict Rosetta to design hydrophobic interfaces because some degree of hydrophilicity is required for solvation in the unbound state, which is important for transient binders.[22] Additionally, the fixed-sequence protein in a one-sided design case may have polar sidechains at the interface; Rosetta must be able to find design solutions that can satisfy the hydrogen bonding potential of these polar atoms.

In 2013, Stranges et al. released a study that collected and analyzed many successful (experimentally shown to behave as predicted) and unsuccessful protein-protein interfaces designed by Rosetta.[23] One of their major takeaways was that the successfully designed interfaces were less polar than both the set of unsuccessful designs and the set of naturally-occurring interfaces used as reference. This suggests that Rosetta is unable to reliably design stable interfaces that have native-like densities of polar sidechains.

A major hindrance in Rosetta's ability to design stable polar interfaces is sampling quality in its high-resolution (meaning all atoms are represented) sampling methods. Rosetta generates a rotamer (sidechain conformation) library for each residue position and uses one of several protocols to determine which rotamer ends up being "placed" at each position.[18] When a residue position is allowed to mutate, the rotamer library for that position will contain sidechain conformations across multiple amino acids. These rotamer libraries can be increased in size by sampling chi (sidechain torsion) angles more finely. Our experiments show that the typical levels of chi-angle sampling are insufficient to sample most hydrogen bonding conformations at native protein-protein interfaces.[24] For this reason, rotamer libraries tend to be too large to sample exhaustively and must rely on stochastic search protocols.[24,25] Stranges et al. suggest that these search protocols have room for improvement with regard to hydrogen bond sampling.[23]

1.2 Opportunities for Innovation

1.2.1 Improved Rotamer Sampling

Stranges et al. had two calls to action in the conclusion of their study: (1) Rosetta's score function needs further development to more accurately model desolvation penalties and (2) Rosetta's sampling techniques need to be better suited to sample hydrogen-bonding partners across in the interface.[23] Figure 1.1 shows a timeline of relevant Rosetta developments since that study was released. As you can see, Rosetta developers have put a considerable amount of work into the first call to action but have not spent as much time working on sampling improvements.[26–28]

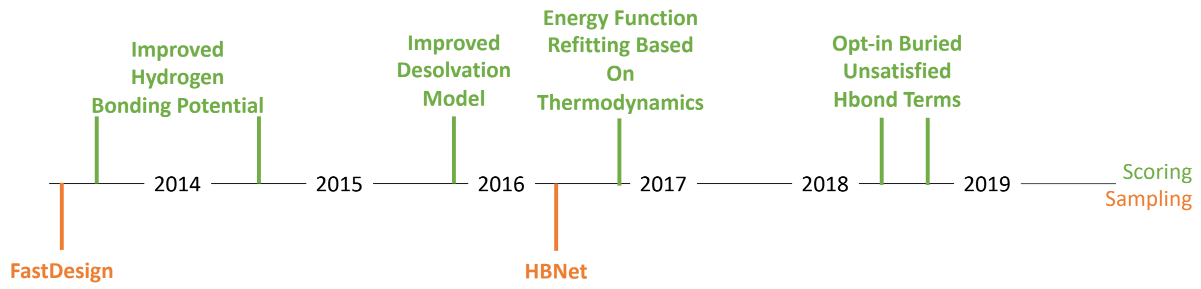

Improved Hydrogen Bonding Potential
Improved Desolvation Model
Energy Function Refitting Based On Thermodynamics
Opt-in Buried Unsatisfied Hbond Terms
2014
2015
2016
2017
2018
2019
Scoring
Sampling
FastDesign
HBNet

Figure 1.1 *Timeline of major Rosetta developments relevant to interface design since the Stranges et al.*[23] *study.*

Only two sampling developments have been made in the same timeframe, and we will see in Chapters 2-4 that both of those developments do not adequately address Rosetta's interface design sampling. The first sampling development, FastDesign,[18] recognized that atomic interactions such as hydrogen bonds are sensitive to small geometric perturbations in the protein's backbone. With this premise, FastDesign intermixes rounds of rotamer substitution with energy minimization using all-atom torsion-angle gradient descent. This allows backbone atoms to make small movements to accommodate mutations. Since hydrogen bonds are so energetically sensitive to small geometric changes, this round of minimization is expected to help optimize hydrogen bond formation for protein design in general. I explore the benefits and pitfalls of FastDesign more in Chapter 3.

The second sampling development, HBNet,[12] prepopulates protein interfaces with hydrogen bond networks prior to rotamer sampling. A subset of residue positions are permanently assigned conformations that are known to hydrogen bond with one another. Then, the rotamer substitution will take place around the fixed (immobile for the sake of preserving conformation) residues in the hydrogen bond network. The goal of HBNet is to explicitly prevent unsatisfied polar atoms at interfaces, with the tradeoff of possible suboptimal rotamer packing. We take a deeper dive into HBNet in Chapter 2.

1.2.2 Improved Backbone Sampling Efficiency

Most protein interface design projects involve backbone sampling. This generally involves docking the protein chains together but can also include backbone-generating protocols like loop-sampling or SEWING (a protocol that builds a protein chain from scratch).[13,19] These design projects consider their task in two phases: (1) backbone sampling and (2) fixed-backbone rotamer sampling. The first phase often generates thousands or millions of backbones (decoys). The second phase is too computationally expensive to be run millions of times, so the user only performs rotamer-sampling on the decoys that are predicted to be the most fruitful. This prediction is done by a “low-resolution” score function, meaning a score function that only evaluates a protein’s backbone (though some low-resolution score functions implicitly model sidechains[25]).

We will see in Chapter 5 that Rosetta’s current low-resolution score functions given suboptimal results, meaning that some fruitful interface decoys may be discarded between the backbone sampling phase and the rotamer sampling phase by being a false negative. By improving these score functions, we can improve the efficiency at which good protein-protein interfaces can be designed.

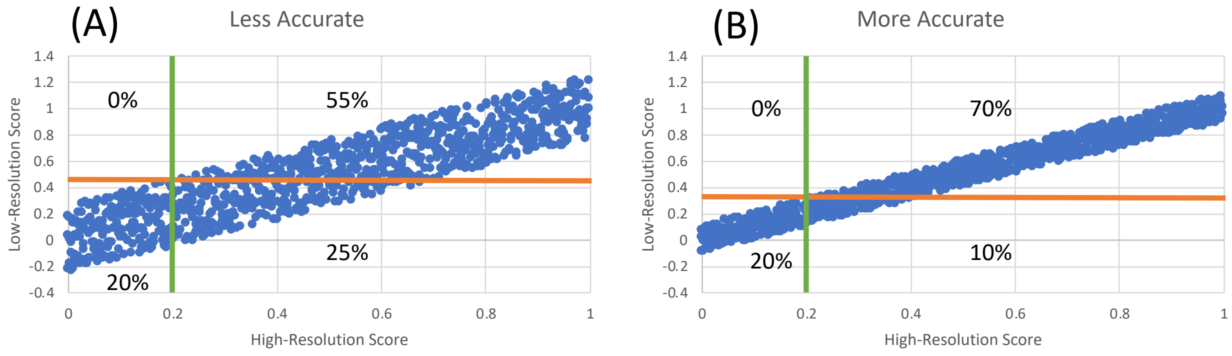


Figure 1.2 *Toy illustration of our idea for improved backbone sampling efficiency. These axes are in made-up, nameless units in which lower numbers are better. (A) Correlation between high- and low-resolution score terms with a less accurate low-resolution score function. (B) Silimar correlation but with a more accurate low-resolution score function. Data points to the left of the green line are considered fruitful. Data points below the orange line must pass the low-resolution filter in order for all of the fruitful points to pass. Both figures show the percentage of total points in each quadrant.*

Figure 1.2 shows a toy example that attempts to illustrate this concept by contriving two sets of data. Subfigure 1.2A shows correlation between high-resolution scores and corresponding low-resolution scores with some degree of misprediction noise. Subfigure 1.2B shows an equivalent data set in which the low-resolution score function has a smaller degree of misprediction noise. Now, imagine that a user wants to extract all of the fruitful decoys and sets their cutoff to be 0.2 (less positive is considered better). The actual high-resolution score is not known until the expensive all-atom sampling is complete, so the user must then determine a cutoff using the low-resolution score function. Figure 1.2 shows in green the intended high-resolution cutoff line and in orange shows the strictest possible low-resolution cutoff that still includes all of the fruitful points. As you can see by the percentage labels, the more accurate low-resolution score function decreases the false positive count (from 25% to 10% of the total runs in this contrived example). The expensive high-resolution sampling would take place on all points below the orange lines, so the lower false positive count would save a significant amount of computer time (33% speedup in this case).

1.3 Chapter Organization and Reading Advice

Chapters 2 and 3 describe implementation details for two of our novel developments in great depth. Chapter 4 broadly summarizes these developments and benchmarks their role in protein interface design as a whole, namely the problem outlined in section 1.2.1. For this reason, it may be beneficial to read Chapter 4 first. Then you can decide to read the other chapters or skim/skip them.

Chapter 5 is somewhat unrelated to the others and describes an approach to the problem outlined in section 1.2.2. My contribution to the work in Chapter 5 is complete, however the project as a whole is still a work in progress. Specifically, my lab mates and collaborators will be benchmarking the tool outlined in Chapter 5 in real-world design applications. Hopefully any unanswered questions will be answered in future publications.

1.4 Concessions

This dissertation presents new computational techniques and shows that they are improvements relative to various computational metrics. Because these metrics are man-made models that only mimic nature, the scopes of my conclusions do not exceed beyond the protein modeling framework. It is the responsibility of experimental biochemists to determine if the methods presented here actually result in better proteins when expressed in the real world.

REFERENCES

1. Kuhlman, B. & Bradley, P. Advances in protein structure prediction and design. *Nature Reviews Molecular Cell Biology* vol. 20 681–697 (2019).

2. Wang, C. *et al*. Current Strategies and Applications for Precision Drug Design. *Front. Pharmacol.* **9**, 787 (2018).

3. Watanabe, M., Matsuzawa, T. & Yaoi, K. Rational protein design for thermostabilization of glycoside hydrolases based on structural analysis. *Appl. Microbiol. Biotechnol.* (2018) doi:10.1007/s00253-018-9288-7.

4. Dahiyat, B. I. & Mayo, S. L. Protein design automation. *Protein Sci.* **5**, 895–903.

5. Zeng, J. Mini-Review: Computational Structure-Based Design of Inhibitors that Target Protein Surfaces. *Comb. Chem. High Throughput Screen.* **3**, (2000).

6. Sammond, D. W., Eletr, Z. M., Purbeck, C. & Kuhlman, B. Computational design of second-site suppressor mutations at protein-protein interfaces. *Proteins Struct. Funct. Bioinforma.* **78**, 1055–1065 (2010).

7. Kortemme, T. & Baker, D. Computational design of protein-protein interactions. *Curr. Opin. Chem. Biol.* **8**, 91–97 (2004).

8. Joachimiak, L. A., Kortemme, T., Stoddard, B. L. & Baker, D. Computational Design of a New Hydrogen Bond Network and at Least a 300-fold Specificity Switch at a Protein–Protein Interface. *J. Mol. Biol.* **361**, 195–208 (2006).

9. Kapp, G. T. *et al*. Control of protein signaling using a computationally designed GTPase/GEF orthogonal pair. *Proc. Natl. Acad. Sci.* **109**, 5277–5282 (2012).

10. Mandell, D. J. & Kortemme, T. Computer-aided design of functional protein interactions. *Nat. Chem. Biol.* **5**, 797–807 (2009).

11. Leaver-Fay, A. *et al*. Rosetta3: An Object-Oriented Software Suite for the Simulation and Design of Macromolecules. *Methods Enzymol.* **487**, 545–574 (2011).

12. Boyken, S. *et al*. De novo design of protein homo-oligomers with modular hydrogen bond network-mediated specificty. *Science* **399**, 69–72 (2016).

13. Guffy, S. L., Teets, F. D., Langlois, M. I. & Kuhlman, B. Protocols for Requirement-Driven Protein Design in the Rosetta Modeling Program. *J. Chem. Inf. Model.* **58**, 895–901 (2018).

14. Tinberg, C. E. & Khare, S. D. Computational Design of Ligand Binding Proteins. in *Methods in molecular biology (Clifton, N.J.)* vol. 1529 363–373 (2017).

15. Stranges, P. B. & Kuhlman, B. A comparison of successful and failed protein interface designs highlights the challenges of designing buried hydrogen bonds. **22**, 74–82 (2013).

16. Leaver-Fay, A. *et al*. Computationally Designed Bispecific Antibodies using Negative State Repertoires. *Structure* **24**, 641–651 (2016).

17. Fleishman, S. J. *et al*. Computational design of proteins targeting the conserved stem region of influenza hemagglutinin. *Science* **332**, 816–21 (2011).

18. Tyka, M. D. *et al*. Alternate states of proteins revealed by detailed energy landscape mapping. *J Mol Biol* **405**, 607–618 (2011).

19. Kaufmann, K. W., Lemmon, G. H., Deluca, S. L., Sheehan, J. H. & Meiler, J. Practically Useful: What the ROSETTA Protein Modeling Suite Can Do for You. doi:10.1021/bi902153g.

20. Xu, D., Tsai, C. J. & Nussinov, R. Hydrogen bonds and salt bridges across protein-protein interfaces. *Protein Eng*. **10**, 999–1012 (1997).

21. Fleming, P. J. & Rose, G. D. Do all backbone polar groups in proteins form hydrogen bonds? *Protein Sci*. **14**, 1911–7 (2005).

22. Chandler, D. Interfaces and the driving force of hydrophobic assembly. *Nature* **437**, 640–647 (2005).

23. Stranges, P. B. & Kuhlman, B. A comparison of successful and failed protein interface designs highlights the challenges of designing buried hydrogen bonds. *Protein Sci*. **22**, 74–82 (2013).

24. Maguire, J. B., Boyken, S. E., Baker, D. & Kuhlman, B. Rapid Sampling of Hydrogen Bond Networks for Computational Protein Design. *J. Chem. Theory Comput*. (2018) doi:10.1021/acs.jctc.8b00033.

25. Kuhlman, B. *et al*. Design of a Novel Globular Protein Fold with Atomic-Level Accuracy. *Science (80-.)*. **302**, 1364–1368 (2003).

26. Leaver-Fay, A. *et al*. *Scientific benchmarks for guiding macromolecular energy function improvement*. *Methods in Enzymology* vol. 523 (Elsevier Inc., 2013).

27. O'Meara, M. J. *et al*. Combined covalent-electrostatic model of hydrogen bonding improves structure prediction with Rosetta. *J. Chem. Theory Comput*. **11**, 609–22 (2015).

28. Alford, R. F. *et al*. The Rosetta All-Atom Energy Function for Macromolecular Modeling and Design. *J. Chem. Theory Comput.* **13**, 3031–3048 (2017).

CHAPTER 2: Rapid Sampling of Hydrogen Bond Networks for Computational Protein Design[1]

2.1 Introduction

Hydrogen bonds are essential for specifying biomolecular structure, and proteins often employ extensive networks of hydrogen bonds to preorganize catalytic active sites[2–5], mediate interaction specificity[6,7], and achieve structure and function with a high level of cooperativity.[8–10] Hydrogen bond networks at protein−protein interfaces help overcome desolvation costs associated with binding while providing polar groups that contribute to the solubility of the unbound monomers. The ability to accurately create new hydrogen bond networks is critical for many problems in protein design, and rational design approaches have successfully achieved networks that specify membrane protein interactions[11] and the coordination of functional metal cofactors;[12–16] however, developing general computational methods for this problem has been challenging.[17] This is in part because hydrogen bond strength is very sensitive to small perturbations in the relative positions of the atoms forming the hydrogen bond.[18,19] Designing buried hydrogen bonds at protein interfaces has been particularly difficult.[20,21]

A key challenge in designing hydrogen bond networks is ensuring that each polar group in a protein or complex has a hydrogen bond partner or is exposed to solvent. It has been estimated that the energetic cost of burying a hydrogen-bond donor or acceptor that does not have a hydrogen bond partner ("unsatisfied") is 5−6 kcal/mol,[22] which is comparable to the total free energy of unfolding for some proteins. The energy functions that are typically used for computational protein design,[23,24] including the Rosetta Energy Function,[25] are expressed as the sum of pairwise energies, which is important for computational efficiency and algorithmic compatibility.[26] However, networks of hydrogen bonds that span multiple residues are inherently not pairwise decomposable, and evaluating burial and hydrogen bond satisfaction cannot be achieved in a pairwise manner. Hence, conventional protein design algorithms are not well-suited for capturing and evaluating satisfied hydrogen bond networks.

Recently, to enable the computational design of hydrogen bond networks, we developed a sampling protocol (HBNet[27]) in the Rosetta software package[28] that explicitly searches through sequence space and side chain conformational space (rotamers) to find sets of amino acids that can form self-contained hydrogen bond networks. To maximize the number of potential networks that are identified for a given backbone conformation and set of residues, HBNet enumerates through all possible closed networks that can be created with a given rotamer library. We define a "closed network" to be one in which every buried polar group has a hydrogen bond partner. HBNet was experimentally validated by using it to design highly symmetric networks in the center of de novo designed coiled coils.[27] Since these initial results, we have begun to apply HBNet to other design problems, and we have found that it scales poorly to larger systems, especially cases where symmetry cannot be used to reduce search space. For larger rotamer libraries or larger numbers of residue positions, we observe that the exhaustive search employed by HBNet often does not complete after several hours, which precludes its use in design pipelines that also involve backbone sampling and docking. Here, we introduce a new Monte Carlo-based algorithm (MC HBNet) that makes it possible to rapidly sample and design viable hydrogen bond networks for larger design problems.

The MC HBnet protocol begins by building a graph in which each node in the graph represents a potential side chain conformation (rotamer) for an amino acid at a specified sequence position, and an edge is drawn between two nodes if a hydrogen bond is formed between the two rotamers. Hydrogen bond networks are then assembled by stochastically traversing the graph and outputting networks that do not leave any buried polar group without a hydrogen bond partner. We show that MC HBNet is able to recapitulate the networks of native protein–protein interfaces and that it can be robustly used with large rotamer libraries. Because MC HBNet can be used with a finer degree of side chain sampling than HBNet, we also show that it can find more favorable networks than HBNet in substantially shorter runtimes. These improvements allow explicit hydrogen bond network design to be incorporated into more complex, multistage protocols such as de novo interface design and enzyme design and are general strategies that can be readily incorporated into modeling packages other than Rosetta.

2.2 Methods

2.2.1 Side Chain Sampling and Identification of Hydrogen Bonds

MC HBNet begins by examining residue pairs and identifying which amino acid mutations and side chain rotamers at the two positions will allow the formation of one of more hydrogen bonds between the residues. All residue pairs within a user-defined set of packable positions are enumerated. The set of amino acids considered at each position are also defined by the user. The protein backbone is held fixed throughout the protocol, and side chain conformations are sampled using a backbone-dependent rotamer library.[29] The user can specify to consider only side chain conformations constructed from the most preferred side chain torsion angles (chi angles) for each rotamer ("base" rotamer), or the rotamer library can be expanded by introducing extra chi sampling. Extra chi sampling builds additional rotamers from base rotamers by varying the side chain's chi angles by an amount determined by statistical measurements of that chi angle's variance in high resolution crystal structures of naturally occurring proteins. The magnitude and frequency of extra chi sampling can be controlled by the user (Table 2.1).

label	chi 1	chi 2	chi 3	chi 4
Ø	0	0	0	0
$\chi 1$	1	0	0	0
$\chi 1 \chi 2$	1	1	0	0
$\chi 1 \chi 2 \chi 3$	1	1	1	0
$\chi 1 \chi 2 \chi 3 \chi 4$	1	1	1	1

Table 2.1 *Definitions of Extra-Chi Sampling Levels. A 0 in any column means that there were no extra samples for that chi angle. A 1 in any column means that chi angle had extra samples at ±1 standard deviation defined by the Dunbrack backbone dependent rotamer library.*[29]

After side chain coordinates are calculated for the rotamers being considered at each designable position, each rotamer pair from all the residue pairs are examined to determine if a hydrogen bond is being formed between the rotamers. Hydrogen bonds are detected using Rosetta's standard hydrogen bonding potential, which depends on the distance and relative orientation of the donor and acceptor groups.[18,30] Hydrogen bonds typically score between –0.5 and −1.5 Rosetta Energy Units (REU). For most of this work, we consider only interactions with an energy less than –0.5 as a hydrogen bond. When a hydrogen bond is detected between two rotamers, this information is saved in an interaction graph that is used during the sampling protocol. MC HBNet uses a new data structure called HbondGraph that includes nodes for each rotamer, as well as atom-level information for each hydrogen bond, enabling more efficient organization and lookup as compared to the graph used by the original implementation of HBNet, which has been described previously.[27]

2.2.2 HBondGraph Data Structure

HBondGraph creates a node for every candidate rotamer at each position (Figure 2.1). An edge is created between every pair of nodes whose corresponding rotamers form a hydrogen bond. Additionally, nodes store information about which other nodes in the HbondGraph are incompatible due to steric clashes between their respective rotamers. A hydrogen bond network can be defined as a set of nodes that form a connected component in the HbondGraph without having any two nodes that represent rotamers that clash or occupy the same residue position.

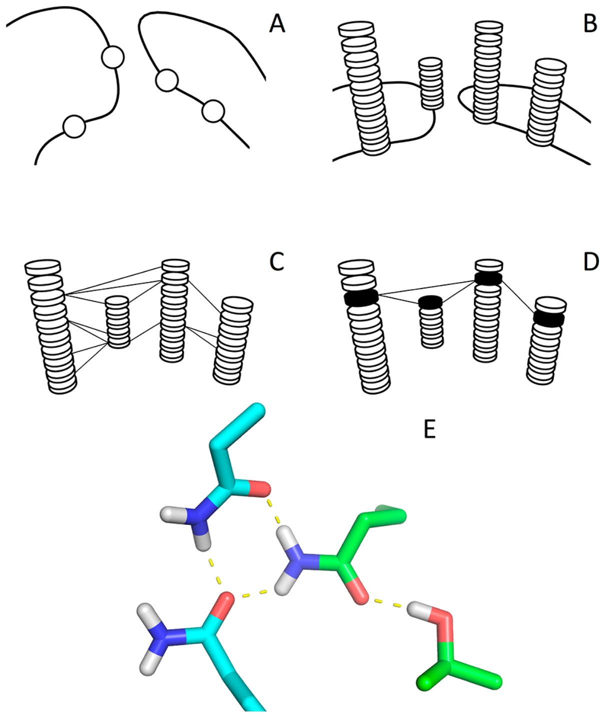


Figure 2.1 *HBondGraph. (A) MC HBNet identifies every residue position that is being designed or repacked. (B) Each position is expanded into an array of graph nodes, one node for every side chain conformation being considered at that position. (C) Graph edges are created between every pair of nodes that form a hydrogen bond. (D) An example of what a hydrogen bond network looks like in this data structure and (E) a possible hydrogen bond network that this example might represent (cyan side chains are part of a different chain than green side chains).*

Each node in the HbondGraph keeps track of every side chain polar atom index in its respective rotamer. MC HBNet strips this atom information for polar atoms that are already satisfied by the background (either implicitly by solvent exposure or explicitly by hydrogen bonds to the backbone or nonpackable side chains, which are held fixed; hydrogen bonds from side chain atoms to backbone atoms are scored and taken into account when evaluating satisfaction). Combining the individual lists of atoms from each node in the network produces an ad hoc checklist of atoms that need to be satisfied for a network to be accepted. For each network, MC HBNet tracks satisfaction by storing a list of all heavy (non-hydrogen) polar atoms that are buried and not satisfied ("heavy unsat"); if a heavy unsat becomes satisfied during network growth, that atom is removed from the list. Satisfaction can be rapidly evaluated because each edge in the HbondGraph stores the atom indices for the acceptor atom, donor atom, and hydrogen atom for every hydrogen bond represented by the edge (there may be multiple hydrogen bonds represented by one edge because a single pair of rotamers can only be connected by one edge but may form multiple hydrogen bonds).

2.2.3 Monte Carlo HbondGraph Traversal

The MC HBNet sampling protocol is composed of a user-defined number of trajectories. Each trajectory begins by randomly selecting a "seed" edge from the HbondGraph, and networks are grown stochastically by adding adjacent edges that lead to compatible nodes. Seed edges can be selected based on predefined starting criteria. An example starting criterion is HBNetStapleInterface,[27] which searches for networks that span across a protein–protein interface; when used with MC HBNet, the HBNetStapleInterface protocol requires that all seed edges must contain at least one node position that is at the interface. Efficiency is improved by restricting sampling to only start at seed edges relevant to the task at hand. Starting criteria can be specified by the user to customize the search for different design scenarios.

After the seed is selected, MC HBNet identifies all of the "candidate" nodes in the HbondGraph that are adjacent to either of the seed's nodes but do not conflict either through steric clashing or sharing a residue position (Figure 2.1). For each of these candidates, MC HBNet counts the number of HbondGraph nodes ("children") adjacent to the candidate that are compatible with both of the two seed nodes. The relative probability of selecting a candidate to be added to the network is proportional to the number of its compatible children plus one. One candidate is stochastically selected to be added to the network, and the network is registered as a result if it is determined to be absent of heavy unsats. This process is repeated until there are no more adjacent nodes in the HbondGraph that are compatible with every node in the network.

The process described above defines a single MC HBNet trajectory and will be repeated a user-defined number of times (the default trajectory setting is 10^4). MC HBNet trajectories are made faster by keeping track of the heavy unsatisfied polar atoms as the network grows. If the network has any heavy unsats at any point within a trajectory, the MC HBNet sampling protocol will shift its focus to only consider candidate nodes whose addition would result in the satisfaction of a heavy unsat. The trajectory is brought to a premature end if the HbondGraph contains no nodes that can achieve this task. This implementation is represented by the λ term in eq 1 and prevents MC HBNet from wasting time by exploring sample space where finding a fully satisfied network is impossible.

$$f(c_i) = \lambda(\beta + 1) \qquad \text{(Eq 2.1)}$$

Equation 2.1 describes $f(c_i)$ as the relative probability of adding candidate c_i (all candidate nodes are shown in blue in Figure 2.2) to the network, and β is the number of compatible children

c_i has. λ is 0 if all of the side chain polar atoms in c_i's root node (shown in black in Figure 2.2) are satisfied and there are nodes with unsatisfied side chain polar atoms in the current network; otherwise, λ is 1.

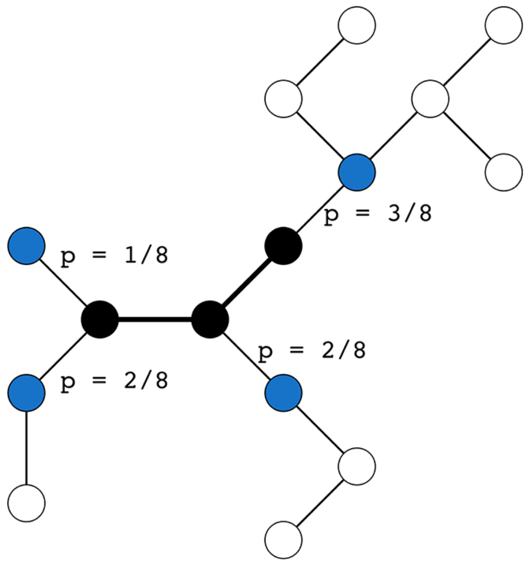


Figure 2.2 *Monte Carlo growth of a network. Residues that are already part of the network are shown in black. Candidate residues are shown in blue, and downstream nodes are shown in white. All nodes and edges exist in the HBondGraph. Blue and white nodes may occur at the same residue positions as other blue and white nodes, but none may occur at the same residue position as a black node. Edges to candidate nodes are labeled with their probability (p) of being added to the network in the next round of Monte Carlo growth.*

Additionally, MC HBNet will not register a network as a result if it contains any heavy-atom donors or acceptors that are buried and unsatisfied. This check reduces MC HBNet's runtime by approximately 40% by creating fewer false positives that need to be filtered out by a more computationally expensive satisfaction check that occurs at the end of the protocol, before networks are output. After all of the trajectories have completed, networks are ranked and prepared for output.

2.2.4 Ranking Results and Output

Networks are filtered and sorted, eliminating those that are redundant in primary amino acid sequence past a user defined threshold. Networks that contain at least one heavy (non-hydrogen) buried polar atom that is not either donating or accepting in a hydrogen bond are also eliminated. Buried polar hydrogen atoms that do not participate in hydrogen bonds are allowed but incur a penalty during sorting. Hydroxyl groups are only required to either donate or accept, but not both, to be considered satisfied (consistent with what is observed in experimentally determined structures[31]). Optionally, one can require that hydroxyl groups donate in order to be considered satisfied. Hydrogen bonds to the backbone are taken into account when evaluating satisfaction; native proteins often make use of hydrogen bonds from side chains to backbone atoms to preorganize structure, and networks that can extend to the backbone are captured by the protocols we present here. Users can also eliminate networks based on a custom criterion (e.g., minimum number of residues in the network, or number of intermolecular hydrogen bonds). The remaining hydrogen bond networks are sorted and ranked first by the number of buried unsatisfied polar hydrogen atoms (Num_Unsat_Hpol), then saturation, then HBNet Score.

2.2.4.1 Saturation

We have previously referred to an early version of this metric as "connectivity," and it was shown that highly connected (saturated) networks were in close agreement with experimentally determined structures, whereas less connected networks were more easily displaced by water molecules.[27] We define saturation as the fraction of total hydrogen bonding capacity (given the polar atoms that comprise the network) that is met by the actual hydrogen bonds of the network. Higher values are better, with 1.0 implying that a network has reached its full hydrogen bond capacity. For every side chain in the network, each polar hydrogen atom on that side chain contributes 1 point and each lone pair contributes 1 point. Only one of the two lone pairs on a hydroxyl oxygen atom contributes a point because it is common for a hydroxyl to be an acceptor to only one hydrogen bond.[31] Saturation is calculated by dividing the sum of the points of all polar side chain atoms in the network by the sum of the points of all polar side chain atoms in the network residues (including atoms that do not participate in network hydrogen bonds). Saturation values can potentially be larger than 1.0 in the case of a hydroxyl participating in three hydrogen bonds or the case of bifurcated hydrogen bonds.

2.2.4.2 HBNet Score

HBNet Score is used to further discriminate between networks that are identical in Num_Unsat_Hpol and saturation by evaluating their energies using the full Rosetta energy function[25] within the same context: the network residues are placed onto a common "background" structure, which is the input structure with all packable residues mutated to alanine (except for any existing Gly, Pro, or disulfides, which are kept). HBNet Score is the difference in energy between the background structure with and without the network residues placed, normalized by the number of residues in the network.

2.2.4.3 Output

Once filtered and sorted, the networks are iteratively placed onto the input structure and reported in order of ranking. During downstream design constraints ensure that the hydrogen bonds of the network are maintained; the assumption is that downstream steps will optimize the space around the hydrogen bond network residues. Users can opt to combine compatible networks on the same output structure. Once the output structures are returned, any other part of Rosetta can be called to perform further design and analysis, or the structures can be output to disk.

2.2.4.4 Burial Calculations

Determining which polar atoms in the networks are buried versus solvent-exposed is challenging because the space around the hydrogen bond networks is often not yet designed, leaving large voids. The original implementation of HBNet used solvent-accessible surface area (SASA)[32,33] calculations with an increased probe radius.[27] In its current form, burial is precomputed by classifying each residue position as buried or not based on the number of neighboring residue positions that fall within a cone around the vector between its Cα and Cβ atoms;[34] this approach is advantageous because the precomputation is faster than the SASA calculations, and it is consistent for each input backbone, yielding the same classification independent of amino acid sequence and side chain conformation.

2.2.5 Benchmarks and Analysis

2.2.5.1 Native Network Recovery

A library of native protein crystal structures was generated by providing the Pisces web server[35,36] with the following conditions: sequence percentage identity ≤ 60; resolution ≤ 2.0 Å; R-factor ≤ 0.3; sequence length 40−10,000; non-X-ray entries excluded; CA-only entries excluded; cull PDB by entry; cull chains within entries set to “No.” This library was pruned to include only structures that contain at least one protein−protein interface. HBNet was used to generate a list of unique native hydrogen bond networks within this pruned library by considering only native rotamers in each structure. This list was filtered to remove networks with at least one side chain that had a heavy atom with a B factor greater than 40 Å^2. A total of 2776 networks met these criteria.

For every network in this list, we identified every hydrogen bond that had an energy ≤ –0.5 REU using Rosetta's ref2015 score function. For every extra-chi sampling level (Table 2.1), we checked to see if it produced a combination of side chains that rebuilt the native network in such a way in which every native hydrogen bond with an energy ≤ –0.5 REU was simultaneously present with an energy ≤ –0.4 REU. If all of the hydrogen bonds in the network could be simultaneously sampled, the network was deemed to be recovered. This was repeated for all 2776 networks and with the extra-chi sampling levels displayed in Table 2.1.

2.2.5.2 Network Design Benchmarks.

Four "motivating" design scenarios were chosen to compare the performances of HBNet and MC HBNet. These scenarios were chosen because they are similar to previous experiments we have run where HBNet did not perform adequately, hence motivating the development of the Monte Carlo protocol: (1) Small interface, one-sided design (PDB code 1YRK): All residue positions of the first chain were designed and all positions on the other chain were set to repack only (amino acid sequence fixed but rotamer conformations sampled), for a total of 40 packable positions. (2) Medium interface one-sided design (PDB code 1DPJ): All residue positions of the first chain were designed and all positions on the other chain were set to repack only, for a total of 121 packable positions. (3) Large interface one-sided design (PDB code 1GK9): All residue positions of the first chain were designed and all positions on the other chain were set to repack only, for a total of 342 packable positions. (4) Small helical bundle monomer (PDB code 3U3B chain A): The 23 buried residue positions were designed and the remaining residue positions were set to repack only. In all cases, all polar amino acid types were considered at designable residue positions, and the input files and scripts needed to run these benchmarks are provided in the Supporting Information Methods. It should be noted that in actual design scenarios, it can be advantageous to be more restrictive regarding which residue positions are designable and which polar amino acid types are allowed at certain positions.

Additionally, we selected four previously published HBNet designs in order to explore how MC HBNet behaves on design scenarios where HBNet has had proven success: PDB code 5J0K (symmetric homodimer), PDB code 5J10 (symmetric homodimer), PDB code 5J0H (symmetric homotrimer), and PDB code 5IZS (symmetric homotrimer). Both HBNet and MC HBNet were run on these design scenarios with sampling levels Ø, $\chi1$, and $\chi1\chi2$ (scripts included in the Supporting Information). MC HBNet was run with trajectory counts of 10^3, 10^4, 10^5, and 10^6 in order for us to explore the amount of sampling that is required to find high-quality networks for the various cases.

We measured the CPU time, peak memory usage, and the average HBNet statistics for the top 10 networks reported for each run. Benchmarks were measured on the Longleaf cluster at the University of North Carolina at Chapel Hill, using Intel Xeon E5-2680 v4 @ 2.40 GHz CPUs. Due to the ability for most runs to finish within a few hours and HBNet's tendency to take weeks to run if given too large of a design scenario, we declared a 24-hour runtime limit. This limit was not applied to the symmetric reruns because they have previously been shown to run in a reasonable amount of time under these conditions. Additionally, this benchmark was run using sampling level Ø on 591 one-sided interface design cases including the three mentioned previously. The only metrics we tracked were the runtimes for HBNet and MC HBNet (using 10^4 trajectories).

2.3 Results and Discussion

2.3.1 Benefits of Extra Chi Sampling

The previously developed HBNet protocol becomes dramatically slower (see below for case examples) when a larger rotamer library (i.e., more chi torsion angle sampling) is used during the design process. Because of the geometric sensitivity of hydrogen bonding, small changes to chi angles can result in substantial differences in the number of hydrogen bonds that can be made between rotamers, especially for longer side chains, for which lever-arm effects can lead to large changes in polar atom position. Thus, increasing chi sampling is generally expected to lead to more hydrogen bonds from which to sample. In order to quantitatively assess the need to be able to handle larger amounts of extra chi sampling, we measured the effect of extra chi sampling on hydrogen bond network sampling. We collected structures for 2776 native hydrogen bond networks at protein–protein interfaces from the PDB. For each network, we rebuilt the amino acid side chains using Rosetta's pool of rotamers and measured the fraction that could be sampled for each extra chi sampling level defined in Table 2.1. If every hydrogen bond in the native network could be simultaneously sampled, then the network was deemed "recoverable" for that chi sampling level. We evaluated every combination of rotamers for the residue positions that create the network, so this search was not compromised by stochasticity. As expected, Figure 2.3 shows the network recovery rate increase as the extra chi sampling level increases. Sampling level $\chi 1 \chi 2$ can sample more than three times the fraction of native networks than can be sampled with no extra chi sampling (Ø).

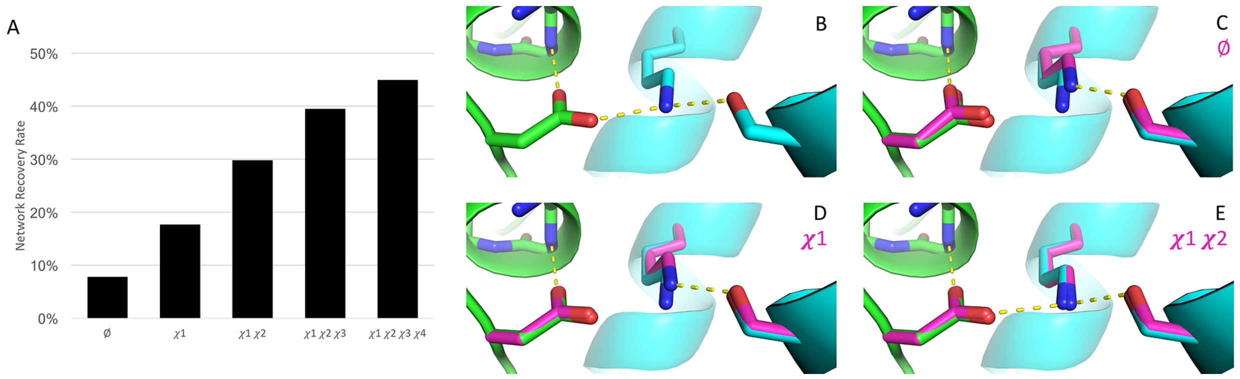


Figure 2.3 *Native networks: impact of extra chi sampling. (A) Fraction of native networks that were sampled with different extra chi sampling levels (levels defined in Table 2.1). Chi sampling level increases from left to right. (B) An example native hydrogen bond network that is recovered at the χ1χ2 extra chi sampling level but not at χ1 or Ø. (C–E) The lowest-RMSD rotamers (magenta) for the native network using the (C) Ø sampling level, (D) χ1 sampling level, and (E) χ1χ2 sampling level.*

Similarly, small changes to the backbone can also propagate to substantial changes in side chain hydrogen bonding geometry and the possible networks that can be generated, and increased chi angle sampling can potentially compensate for these changes. Trajectories from Rosetta's Backrub protocol,[37] which incorporates small degrees of flexibility to generate conformational ensembles, illustrate this concept (Figure 2.S1). Running MC HBNet on this ensemble of backbones shows that backbone perturbations of as little as ~0.1 RMSD can affect the networks that can be captured (Figure 2.S1, middle), and extra chi angle sampling can recover some of the networks that are missed due to these backbone perturbations (Figure 2.S1, bottom). The ability to identify potential networks, even when the backbone is not in the most favorable conformation, will aid the search for low energy design models when performing design protocols that incorporate backbone sampling along with sequence design.

2.3.2 New Design Cases Enabled by MC HBNet

Many important design problems, for instance designing proteins to bind therapeutic targets, involve large asymmetric interfaces. Our initial attempts to design such cases using the original implementation of HBNet resulted in runtimes that were prohibitively slow. To demonstrate that MC HBNet can address these design cases, we assembled a collection of protein–protein interfaces of various sizes. We first searched for networks using a chi sampling level Ø on 591 protein–protein interfaces and only measured the CPU time consumed by each process (Figure 2.4). The space above the diagonal line represents results where HBNet takes longer to run than MC HBNet. Not only are most points above the line, but the distance from the line increases as the problem size grows, demonstrating that MC HBNet is faster than HBNet and better equipped to handle large design cases.

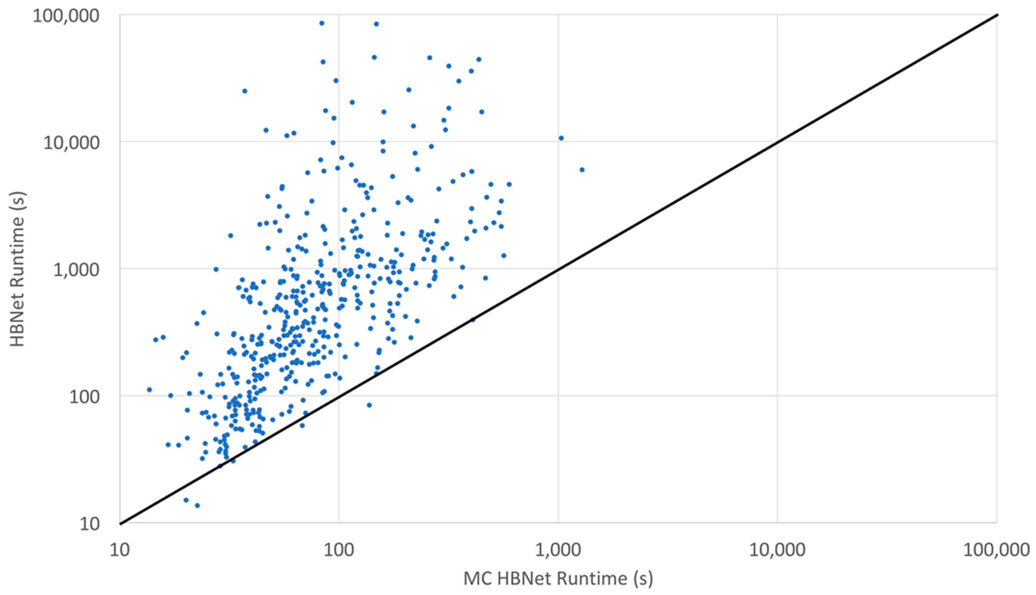


Figure 2.4 *Aggregate data from one-sided interface design benchmarks. The diagonal line represents an equal runtime between the two protocols.*

We next designed networks at three asymmetric protein–protein interfaces of varying sizes as well as the core of a helical bundle monomer (Figure 2.5, Table 2.S2). MC HBNet showed a dramatic speed improvement and a better ability to scale to larger levels of extra chi sampling in all four cases. Many HBNet runs were not able to finish within 24 hours (denoted by asterisk in Figure 2.5), while every MC HBNet run took less than 1 CPU hour. Tables 2.S1–2.S8 also show that MC HBNet is slightly more memory efficient than HBNet for a given extra chi sampling level. All of the top networks reported had 0 unsatisfied polar atoms (which is the first metric used to sort results), meaning that saturation was the primary determinant in the ranking of the networks and assessing network quality. Figure 2.5 plots the average saturation for the 10 best results reported by Rosetta in the rightmost column. MC HBNet displays the ability to find more networks as a function of time than HBNet (Figure 2.5, middle column) and higher quality networks as a function of time. These improvements were consistent for both the asymmetric interfaces, as well as the monomeric design case.

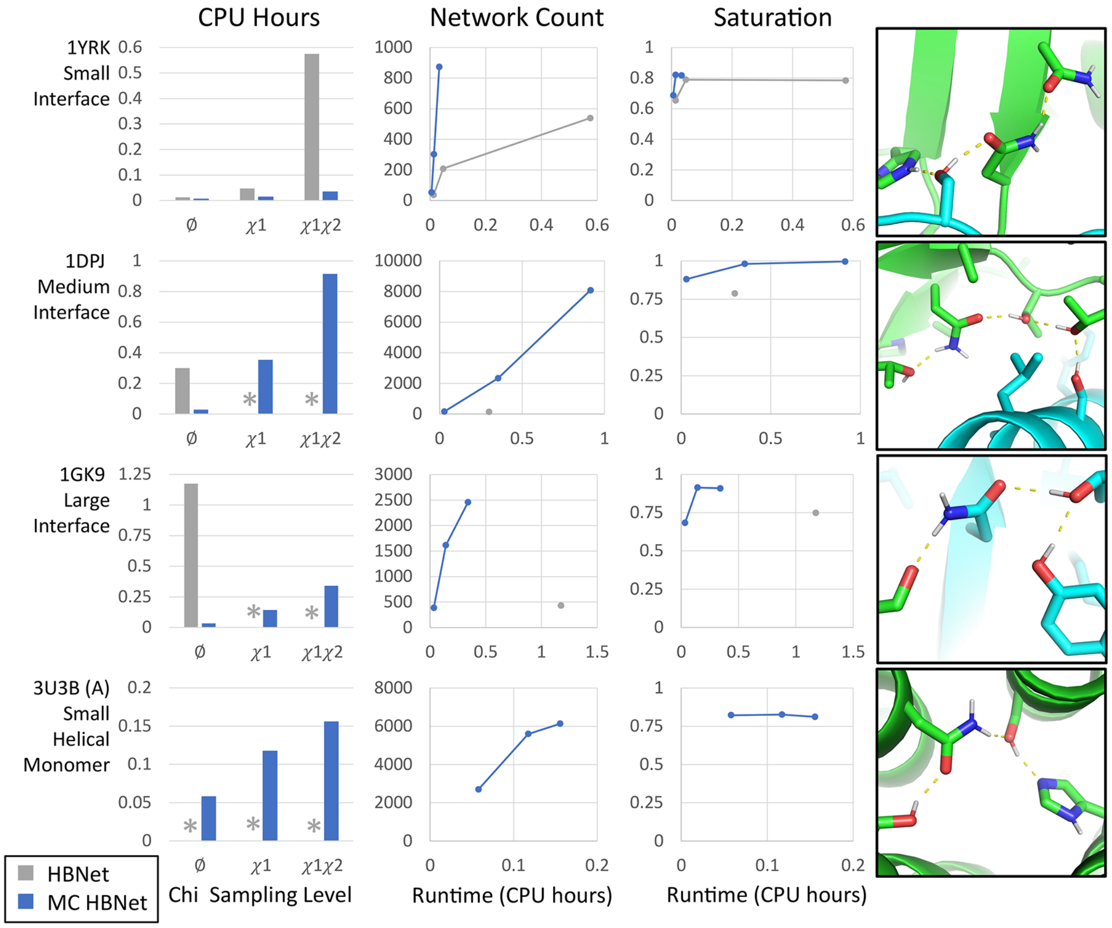


Figure 2.5 *New design problems. Runtime, number of networks found, and saturation for the three interface design benchmarks of various sizes and the small helical monomer. An asterisk in the first column specifies that the protocol did not finish within 24 h. Results of traditional HBNet are shown in gray, and results of the new Monte Carlo protocol are shown in blue. Each case includes a picture of a hand-chosen representative network designed by MC HBNet with χ1χ2.*

HBNet failed to finish exploring the sample space of the small helical monomer design case within our 24-hour time limit, even at the smallest extra chi sampling level, but MC HBNet was able to find thousands of hydrogen bond networks within minutes. Designing hydrogen bond networks into large monomeric structures is challenging, particularly if it is not clear which region of the structure to focus on. The sample space of all possible hydrogen bond networks grows dramatically when the requirement of crossing an interface is removed. Our experience in using HBNet for noninterface designs has often resulted in unreasonably long runtimes. This issue can be partially alleviated by manipulating user-defined options, but it is not always obvious to the user how to implement this effectively. This weakness is not present in MC HBNet's algorithm because the runtime of a Monte Carlo trajectory is not dependent on the size of the sample space.

Surprisingly, MC HBNet can find higher quality networks (defined by saturation) than HBNet using the same extra chi sampling level. This result is not expected to be true when comparing any stochastic protocol with its exhaustive counterpart. The difference is that MC HBNet outputs networks that HBNet cannot; HBNet stores only networks that have grown to completion (ignoring the special case for hydroxyls, see Supporting Information). Network quality can be decreased when residues that contain unsatisfiable polar atoms are added to an already satisfied network. The moniker "satisfied subnetworks" is given to these networks that meet all design requirements and still have the ability to grow. MC HBNet can register satisfied subnetworks as results and continue to grow from them, while HBNet does not by default. In short, HBNet is a complete search of an incomplete sample space while MC HBNet is an incomplete search of a more complete sample space, and the latter appears to be a more effective strategy.

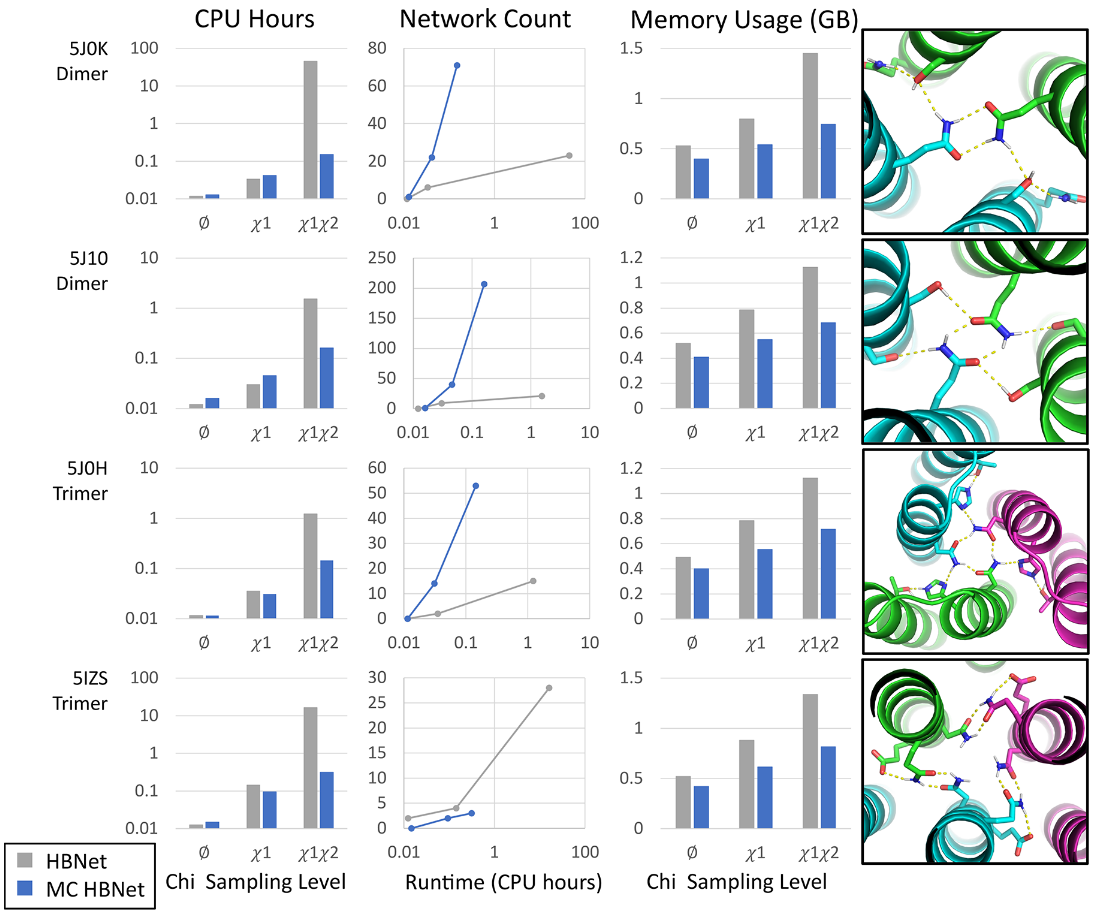


Figure 2.6*. Symmetric interfaces. Runtime, number of networks found, and memory usage for four HBNet designs previously reported.*[27] *Each case is labeled with its PDB code and includes a picture of a hand-chosen representative network designed by MC HBNet with χ1χ2. Results of the original HBNet implementation are shown in gray, and results of the new Monte Carlo protocol are shown in blue.*

2.3.3 Symmetric Homo-Oligomer Benchmarks

We have previously reported success using HBNet to design symmetric homo-oligomers.[27] A handful of these designs were repeated using both HBNet and MC HBNet (Figure 2.6). Unlike with the Monte-Carlo-motivating benchmarks, we defined stricter filters to the designed networks in order to match the original protocol used to create these designs (details and scripts provided in the Supporting Information). We compared the two protocols by the number of networks that meet these strict design criteria. MC HBNet recapitulates the previously validated networks[27] and is able to find more networks as a function of time, and often as a function of chi sampling level, than HBNet.

MC HBNet still outperforms HBNet for the symmetric homo-oligomers when comparing CPU time for a given chi sampling level; however, the difference is milder than with the asymmetric interfaces. This result is likely due to the small problem size of these design cases. Not only are the proteins relatively small, but the presence of symmetry reduces the design space even further. MC HBNet consistently uses less memory than HBNet, in part due to the ability to use the HBondGraph instead of the traditional Rosetta data structures. The full table of results can be found in Tables 2.S5–2.S8; MC HBNet benefits noticeably by increasing the number of Monte Carlo trajectories from the default of 10^4 to 10^5 for these symmetric cases.

2.4 Conclusions

MC HBNet is able to sample hydrogen bond networks faster and more effectively than HBNet. Additionally, MC HBNet can better handle large amounts of candidate rotamers per residue position, which increases the number of hydrogen bond networks that can be identified for a given protein backbone or complex. We have implemented MC HBNet within the Rosetta modeling package, but the sampling strategy, data structures, and network selection criteria described here are general and could be straight forwardly implemented within other computational frameworks.

One of our primary motivations for developing MC HBNet was to create a robust protocol that could be used as part of a larger pipeline aimed at *de novo* interface design. When designing new protein–protein interactions, it is generally not clear *a priori* what will be the most favorable way to dock the proteins against each other. For this reason, interface design protocols generally iterate between sampling alternative docked positions and searching for interface sequences that will stabilize the complex. It is important that the sequence search be rapid and reliably produce low energy solutions so that many alternative docked positions can be sampled. MC HBNet is well suited for this task because it can generally finish in the less than a minute for most interface sizes, and it produces multiple solutions that can be independently carried forward for design calculations to optimize the side chains of the neighboring residues; the computational savings afforded by MC HBNet can be reallocated to employ more computationally expensive protocols (e.g., flexible backbone methods) during downstream design to optimize the remaining interface positions that surround the network. In addition to interface design, this type of protocol should prove useful for designing ligand binding sites and catalytic sites that require hydrogen bond networks to stabilize the ligand or transition state.

2.5 Supplemental Information

2.5.1 Supplementary Methods

MC HBNet and the methods described here are available as part of the Rosetta software package, available from https://www.rosettacommons.org/. Additional documentation for the new methods and software presented here can be found at:

https://www.rosettacommons.org/docs/latest/scripting_documentation/RosettaScripts/Movers/movers_pages/HBNetMover

2.5.1.1 Original HBNet Implementation

The original implementation of HBNet[27] consisted of three steps: 1) An exhaustive search to identify and enumerate networks of hydrogen bonds within a given design space (the pool of sidechain conformations (rotamers) being considered at each designable residue position of an input backbone structure); 2) ranking and sorting of the networks; and 3) iteratively placing each network (or combinations thereof) onto the input structure as starting points for downstream design (a complete working design depends not only on the networks, but on numerous features, including hydrophobic packing around the networks[27]). A limitation of this protocol is that the entire design space must be searched before solutions are returned, resulting in prohibitively long runtimes for many design cases. In these cases, it would be more desirable to enumerate several good solutions in a feasible time frame even if the best solutions are not found.

Most Rosetta protocols use a discrete set of sidechain conformations (rotamers) to model the amino acids at each residue position. The number of rotamers being considered per residue position is a key determinant of Rosetta's ability to sample a given hydrogen bond network. When more rotamers are included, a larger number of favorable networks can be identified. However, including more rotamers means increasing the size of the sample space exponentially, and the runtime of the exhaustive search algorithm scales worse than exponential with respect to the size of the sample space. Furthermore, we show that the networks found in native protein interfaces are often comprised of non-ideal rotamers, suggesting that the inclusion of extra rotamers is beneficial for sampling native-like networks. The steep tradeoff between runtime and the number of rotamers in this implementation means that users must compromise, and very large design systems are not tractable. This behavior also makes it difficult to use HBNet as an intermediate step in a large multi-protocol design pipeline.

The original implementation of HBNet enumerated hydrogen bond networks via recursive traversal of Rosetta’s Interaction Graph data structure. Each node of the Interaction Graph represents a packable residue position and an edge is formed between every pair of nodes that can potentially interact in three-dimensional space as defined by the Rosetta Energy Function. Each edge contains a two-dimensional matrix of interaction energies between all pairs of sidechain rotameric states (rotamers) at those two positions. In HBNet, these matrices are populated with the sum of the sidechain-sidechain hydrogen bond energies and a steric repulsion term (Van der Waals forces). The hydrogen bonding term cannot be greater than zero and the steric repulsion term cannot be less than zero, meaning that a negative value can be interpreted as a hydrogen bond and a positive value can be interpreted as a steric clash.

HBNet performs a recursive depth-first traversal of this graph to enumerate linear hydrogen bond connectivities. By default, the traversal starts at all designable residue positions, but for many cases this is inefficient and it is clear which positions the network search should focus on. Users can explicitly define the starting positions if desired, or in the case of protein interface design (HBNetStapleInterface), the default behavior is for the traversal to start at all interface positions. The recursive traversal continues until one of the following conditions is reached, at which point the current network state is stored as a result:

1. Return to a starting position. Because the traversal is initiated at all starting positions, continuing after reaching another start position results in redundancy; thus, the traversal stops and these initial networks are stored. Starting networks that are compatible (do not clash and share at least one rotamer in common) are combined at a later step.

2. No more hydrogen bonds are found. If a linear network cannot be extended any further, the current state is stored and the traversal stops.
3. Hydroxyl is reached. Since initial publication, a third condition was added that stores network states every time a hydroxyl sidechain is reached. Amino acids with hydroxyl chemical groups (Ser, Thr, Tyr) are excellent at "capping" networks, meaning that it terminates the network in a satisfied state. The reason being that hydroxyl chemical groups can function as both hydrogen bond donors and acceptors, but are only required to do one or the other. Without this condition, many networks continue past hydroxyl sidechains and terminate by condition #2 in an unsatisfied state, hence missing many satisfied solutions.

In this manuscript, we show that by ensuring the inclusion of all satisfied subsets (not only ones that terminate in hydroxyls), the Monte Carlo HBNet implementation finds an increased number of satisfied solutions. The original implementation of HBNet has an option to register all subnetworks (store_subnetworks=true), but this option slows runtime even further to the point that is not feasible in most design cases, and thus is false by default.

After the initial networks are identified, a merging step is performed that identifies all networks that share one or more common rotamers and, after checking for clashes or conflicting residues, combinatorically merges them together into complete networks. This step allows HBNet to create branched networks from the library of linear networks, allowing them to finish growing to their full potential. The reason for this approach is due to limitations imposed by using the Rosetta Interaction Graph. Traditional graph traversal methods are complicated by having to traverse not only the nodes of the graph, but the matrices pointed to by each edge (multiple rotamers per each node, and multiple pairs of rotamers for each edge); and node compatibility cannot be inferred from the edges alone, but rather depends on properties of the three-dimensional coordinates of the sidechain atoms. Thus, the order in which rotamers are added to a network matters during the recursive traversal, restricting downstream possibilities, especially for amino acid types that can participate in more than two hydrogen bonds. Combinatorial sampling of different possibilities and orderings during traversal is complicated due to the way information is stored in the Interaction Graph. The new Monte Carlo-based sampling approach we describe overcomes these complications by using the new HBondGraph instead, and by growing networks stochastically to completion, one at a time, rather than growing networks in parallel during Interaction Graph traversal.

2.5.1.2 MC HBNet runs on Backrub trajectories

MC HBNet runs on Backrub trajectories to demonstrate sensitivity to small backbone perturbations and the ability of the extra Chi sampling afforded by MC HBNet so help mitigate these effects: Small backbone perturbations were generated using Rosetta's Backrub protocol[37], run with the following command on the input pdb 2L6HC3_13, which is the design model that corresponds to PDB ID 5J0H[27]:

```
/path/to/Rosetta/main/source/bin/backrub.linuxgccrelease -movemap movemap -sm_prob 1
- backrub:trajectory -backrub:trajectory_stride 10 -max_atoms 100000 -s
2L6HC3_13_design.pdb
```

Contents of movemap file:

```
RESIDUE * BB JUMP 1 YES
```

MC HBNet was run on the output structures generated from these Backrub trajectories to search for new designed networks using the following Rosetta XML mover definition (networks were restricted to criteria consistent with the experimentally validated design):

No extra Chi sampling (Ø):

```
<HBNetStapleInterface core_selector="core" design_residues="NST" hb_threshold="-0.75"
max_unsat_Hpol="6"       min_core_res="4"       min_helices_contacted_by_network="6"
min_network_size="6"            minimize="false"            monte_carlo_branch="true"
monte_carlo_seed_must_be_fully_buried="true"   name="hbnet_interf"   scorefxn="beta"
show_task="true"              total_num_mc_runs="100000"              verbose="true"
write_network_pdbs="true"/>
```

Extra Chi sampling level $\chi1\chi2$:

```
<HBNetStapleInterface core_selector="core" design_residues="NST" hb_threshold="-0.75"
max_unsat_Hpol="6"       min_core_res="4"       min_helices_contacted_by_network="6"
min_network_size="6"            minimize="false"            monte_carlo_branch="true"
monte_carlo_seed_must_be_fully_buried="true"   name="hbnet_interf"   scorefxn="beta"
show_task="true"  task_operations="ex1_ex2"  total_num_mc_runs="100000"  verbose="true"
write_network_pdbs="true"/>
```

Where “ex1_ex2” Task operation is defined as:

```
<ExtraRotamersGeneric ex1="1" ex2="1" name="ex1_ex2"/>
```

2.5.1.3 Rosetta Script for the three interface "Motivating" cases

```
<ROSETTASCRIPTS>

  <SCOREFXNS>
    <ScoreFunction name="standardfxn" weights="ref2015"/>
  </SCOREFXNS>

  <RESIDUE_SELECTORS>
    <Chain name="chain1" chains="1"/>
    <Chain name="chain2" chains="2"/>
    <InterfaceByVector         name="strict_interface"         grp1_selector="chain1"
grp2_selector="chain2"/>
    <PrimarySequenceNeighborhood name="interface" selector="strict_interface"/>
    <Not name="not_interface" selector="interface"/>
  </RESIDUE_SELECTORS>

  <TASKOPERATIONS>
    <InitializeFromCommandline name="ifc"/>

    <OperateOnResidueSubset name="do_not_design_chain2" selector="chain2">
      <RestrictToRepackingRLT/>
    </OperateOnResidueSubset>

    <OperateOnResidueSubset name="only_pack_interface" selector="not_interface">
      <PreventRepackingRLT/>
    </OperateOnResidueSubset>
  </TASKOPERATIONS>

  <MOVERS>
    <HBNetStapleInterface   name="hbnet"   scorefxn="standardfxn"   minimize="false"
task_operations="ifc,do_not_design_chain2,only_pack_interface"        max_mc_nets="0"
secondary_search="false"   max_replicates="3"   max_replicates_before_unsat_check="5"
max_replicates_before_branch="5"   monte_carlo_branch="true"   write_cst_files="false"
write_network_pdbs="false" total_num_mc_runs="%%runs%%" />

    To run traditional HBNet, set monte_carlo_branch to false. total_num_mc_runs is
set using the "-parser:script_vars" command line option.
  </MOVERS>

  <FILTERS>
    Calculator Filter is used here to create a filter that always fails. This prevents
Rosetta wasting time by outputting many PDB files after running HBNet.

    <CalculatorFilter name="always_fails" threshold="0" equation="k" >
      <Var name="k" value="1" />
    </CalculatorFilter>
  </FILTERS>

  <PROTOCOLS>
    <Add mover="hbnet" filter="always_fails"/>
  </PROTOCOLS>

</ROSETTASCRIPTS>
```

2.5.1.4 Rosetta Script for the two symmetric homodimer cases

```
<ROSETTASCRIPTS>

  <TASKOPERATIONS>
    <InitializeFromCommandline name="init"/>
    <IncludeCurrent name="current"/>
    <LimitAromaChi2 name="arochi" />
    <ExtraRotamersGeneric name="ex1_ex2" ex1="1" ex2="1"/>

    <LayerDesign name="init_layers" layer="other" make_pymol_script="0">
      <TaskLayer>
       <SelectBySASA name="symmetric_inteface_core" state="bound" mode="mc" core="1"
probe_radius="2.0" core_asa="35" surface_asa="45" verbose="1"/>
       <all copy_layer="core" />
       <Helix append="NQSTH"/>
      </TaskLayer>

      <TaskLayer>
       <SelectBySASA   name="symmetric_inteface_surface"   state="bound"   mode="mc"
surface="1" probe_radius="2.0" core_asa="35" surface_asa="45" verbose="1"/>
       <all copy_layer="surface" />
      </TaskLayer>

      <TaskLayer>
       <SelectBySASA   name="symmetric_inteface_boundary"   state="bound"   mode="mc"
boundary="1" probe_radius="2.0" core_asa="35" surface_asa="45" verbose="1"/>
       <all copy_layer="boundary" />
       <Helix exclude="EKRW"/>
      </TaskLayer>
    </LayerDesign>
  </TASKOPERATIONS>

  <MOVERS>
    <DetectSymmetry name="detect_symm" />

    <HBNetStapleInterface          name="hbnet_interf"          hb_threshold="-0.75"
upper_score_limit="3.5"              write_network_pdbs="0"              minimize="0"
min_helices_contacted_by_network="4"     min_network_size="4"     max_unsat_Hpol="2"
max_networks_per_pose="4"           combos="2"           onebody_hb_threshold="-0.3"
task_operations="init,current,arochi,init_layers"          monte_carlo_branch="true"
max_mc_nets="0" total_num_mc_runs="%%runs%%" write_cst_files="false" />

    To run traditional HBNet, set monte_carlo_branch to false. total_num_mc_runs is
set using the “–parser:script_vars” command line option.
  </MOVERS>

  <PROTOCOLS>
    <Add mover_name="detect_symm"/>
    <Add mover_name="hbnet_interf"/>
  </PROTOCOLS>

</ROSETTASCRIPTS>
```

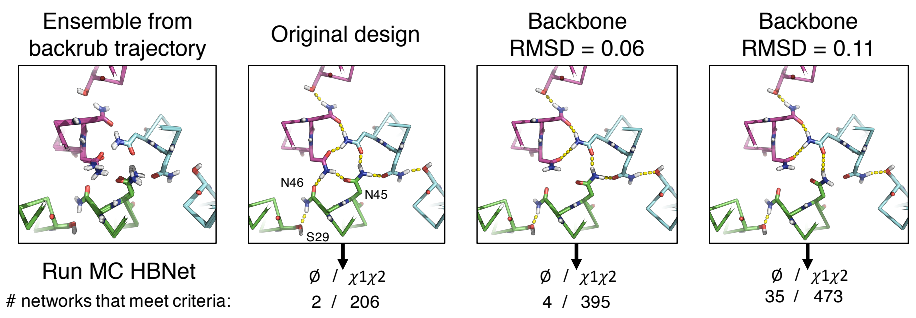
2.5.1.5 Rosetta Script for the two symmetric homotrimer cases

```
<ROSETTASCRIPTS>

  <TASKOPERATIONS>
    <InitializeFromCommandline name="init"/>
    <IncludeCurrent name="current"/>
    <LimitAromaChi2 name="arochi" />
    <ExtraRotamersGeneric name="ex1_ex2" ex1="1" ex2="1"/>

    <LayerDesign name="init_layers" layer="other" make_pymol_script="0">
      <TaskLayer>
       <SelectBySASA name="symmetric_inteface_core" state="bound" mode="mc" core="1"
probe_radius="2.0" core_asa="35" surface_asa="45" verbose="1"/>
       <all copy_layer="core" />
       <Helix append="NQSTH"/>
      </TaskLayer>

      <TaskLayer>
       <SelectBySASA   name="symmetric_inteface_surface"   state="bound"   mode="mc"
surface="1" probe_radius="2.0" core_asa="35" surface_asa="45" verbose="1"/>
       <all copy_layer="surface" />
      </TaskLayer>

      <TaskLayer>
       <SelectBySASA   name="symmetric_inteface_boundary"   state="bound"   mode="mc"
boundary="1" probe_radius="2.0" core_asa="35" surface_asa="45" verbose="1"/>
       <all copy_layer="boundary" />
       <Helix exclude="EKRW"/>
      </TaskLayer>
    </LayerDesign>
  </TASKOPERATIONS>

  <MOVERS>
    <DetectSymmetry name="detect_symm" />

    <HBNetStapleInterface          name="hbnet_interf"          hb_threshold="-0.75"
upper_score_limit="3.5"              write_network_pdbs="0"              minimize="0"
min_helices_contacted_by_network="6"      min_network_size="6"      max_unsat_Hpol="2"
max_networks_per_pose="4"           combos="2"           onebody_hb_threshold="-0.3"
task_operations="init,current,arochi,init_layers"           monte_carlo_branch="true"
max_mc_nets="0" total_num_mc_runs="%%runs%%" write_cst_files="false" />

    To run traditional HBNet, set monte_carlo_branch to false. total_num_mc_runs is
set using the “–parser:script_vars” command line option.
  </MOVERS>

  <PROTOCOLS>
    <Add mover_name="detect_symm"/>
    <Add mover_name="hbnet_interf"/>
  </PROTOCOLS>

</ROSETTASCRIPTS>
```

2.5.1.6 Rosetta Script for the small helical monomer "Motivating" case

```
<ROSETTASCRIPTS>

  <TASKOPERATIONS>
    <InitializeFromCommandline name="ifc"/>
    <ReadResfile name="rrf" filename="resfile"/>
    <IncludeCurrent name="ic"/>
  </TASKOPERATIONS>

  <MOVERS>
    <HBNet    name="hbnet"    task_operations="ifc,rrf,ic"    secondary_search="false"
minimize="false"       max_replicates="3"       max_replicates_before_unsat_check="5"
max_replicates_before_branch="5"       monte_carlo_branch="true"       max_mc_nets="0"
total_num_mc_runs="%%runs%%" write_cst_files="false" write_network_pdbs="false" />

    To run traditional HBNet, set monte_carlo_branch to false. total_num_mc_runs is
set using the "–parser:script_vars" command line option.
  </MOVERS>

  <PROTOCOLS>
    <Add mover="hbnet"/>
  </PROTOCOLS>

</ROSETTASCRIPTS>
```

2.5.2 Supplementary Figures



Example network that can be recapitulated with increased Chi angle sampling:
A_S_22,A_N_23,A_N_52,A_N_53,B_S_22,B_N_52,B_N_53,C_S_22,C_N_52,C_N_53

	HBNet rank	residues	# res	score	Total h-bonds	% saturation	unsat Hpol	Interface h-bonds	Helics contacted
original no extra chi (Ø)	**network not found**								
original ex1ex2 (χ1χ2)	**network not found**								
0053 (RMSD 0.06) no extra chi (Ø)	**Network not found**								
0053 (RMSD 0.06) ex1ex2 (χ1χ2)	**network_44**	A_S_22,A_N_23,A_N_52,A_N_53,B_S_22,B_N_52,B_N_53,C_S_22,C_N_52,C_N_53	10	2.62351	11	0.617647	2	6	6
0101 (RMSD 0.11) no extra chi (Ø)	**network_9**	A_S_22,A_N_23,A_N_52,A_N_53,B_S_22,B_N_52,B_N_53,C_S_22,C_N_52,C_N_53	10	1.73627	10	0.588235	2	6	6
0101 (RMSD 0.11) ex1ex2 (χ1χ2)	**network_51**	A_S_22,A_N_23,A_N_52,A_N_53,B_S_22,B_N_52,B_N_53,C_S_22,C_N_52,C_N_53	10	2.51947	10	0.588235	2	6	6

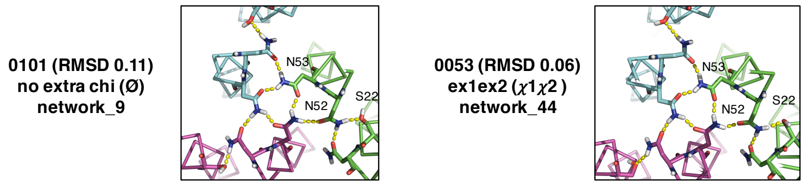


Figure 2.S1 *Small backbone changes affect possible hydrogen bonds and network connectivities: Trimeric design 2L6HC3_13 was perturbed using the Backrub protocol in Rosetta; (top) Outputs from Backrub trajectory illustrating how very small backbone changes can propagate to affect hydrogen bonding; colored by chain, hydrogen bonds shown by yellow dashed lines. (middle) The small backbone perturbations yield substantially different output in MC HBNet design runs to create new networks, using either no extra Chi sampling (Ø), or extra Chi sampling (level (χ1χ2). (bottom) Example showing that extra chi sampling (χ1χ2) can recover some networks that are missed due to the small backbone perturbations and no extra Chi angle sampling. The nomenclature is [chain ID]_[aa type]_[position], using numbering that starts at 1 for each chain; for example, A_S_22 indicates chain A, serine, residue position 22. Together, these results suggest that extra Chi angle sampling, which is now feasible using the new MC HBNet algorithm, can compensate somewhat for small backbone changes that may be difficult to sample explicitly as separate backbone inputs.*

2.5.3 Supplementary Tables

HBNet

Chi Sampling Level		CPU Hours	Memory Usage (GB)	Number of Networks Found	Mean Num_Unsat_Hpol	Mean Saturation	Mean Score
∅		0.01	0.38	37	0	0.65	-0.60
$\chi1$		0.05	0.42	208	0	0.79	2.52
$\chi1\chi2$		0.57	0.56	539	0	0.79	2.13

Monte Carlo HBNet

Chi Sampling Level	Monte Carlo Trajectories	CPU Hours	Memory Usage (GB)	Number of Networks Found	Mean Num_Unsat_Hpol	Mean Saturation	Mean Score
∅	1000	0.01	0.37	41	0	0.67	0.19
∅	10,000	0.01	0.37	56	0	0.72	-0.37
∅	100,000	0.01	0.37	55	0	0.71	-0.33
∅	**1,000,000**	**0.01**	**0.37**	**54**	**0**	**0.69**	**-0.09**
$\chi1$	1000	0.01	0.40	110	0	0.73	-0.07
$\chi1$	10,000	0.01	0.41	223	0	0.81	1.58
$\chi1$	100,000	0.01	0.41	285	0	0.81	1.60
$\chi1$	**1,000,000**	**0.01**	**0.41**	**302**	**0**	**0.82**	**1.79**
$\chi1\chi2$	1000	0.02	0.50	205	0	0.73	0.25
$\chi1\chi2$	10,000	0.02	0.50	462	0	0.79	1.56
$\chi1\chi2$	100,000	0.02	0.51	792	0	0.82	1.85
$\chi1\chi2$	**1,000,000**	**0.03**	**0.56**	**873**	**0**	**0.82**	**1.49**

***Table 2.S1** Data for "Small Interface" case, used to generate Figure 2.5 (MC HBNet data from Figure 2.5 is shown in bold). Mean values were calculated using the top 10 results reported by Rosetta.*

HBNet

Chi Sampling Level		CPU Hours	Memory Usage (GB)	Number of Networks Found	Mean Num_Unsat_Hpol	Mean Saturation	Mean Score
∅		0.30	0.46	137	0	0.79	-0.78
$\chi1$	Did Not Finish						
$\chi1\chi2$	Did Not Finish						

Monte Carlo HBNet

Chi Sampling Level	Monte Carlo Trajectories	CPU Hours	Memory Usage (GB)	Number of Networks Found	Mean Num_Unsat_Hpol	Mean Saturation	Mean Score
∅	1000	0.01	0.45	60	0	0.76	-0.59
∅	10,000	0.02	0.45	124	0	0.88	-0.80
∅	**100,000**	**0.03**	**0.45**	**157**	**0**	**0.88**	**-0.87**
∅	1,000,000	0.03	0.45	171	0	0.88	-0.84
$\chi1$	1000	0.03	0.56	176	0	0.90	-0.70
$\chi1$	10,000	0.08	0.58	762	0	0.97	-0.59
$\chi1$	**100,000**	**0.35**	**0.65**	**2335**	**0**	**0.98**	**-0.52**
$\chi1$	1,000,000	0.95	1.04	6358	0	0.98	-0.74
$\chi1\chi2$	1000	0.07	0.87	351	0	0.82	0.17
$\chi1\chi2$	10,000	0.25	0.92	1778	0	0.97	0.56
$\chi1\chi2$	**100,000**	**0.92**	**1.15**	**8088**	**0**	**1.00**	**0.91**
$\chi1\chi2$	1,000,000	2.60	2.28	30,210	0	1.00	2.60

***Table 2.S2.** Data for "Medium Interface" case, used to generate Figure 2.5 (MC HBNet data from Figure 2.5 is shown in bold). Mean values were calculated using the top 10 results reported by Rosetta*

HBNet

Chi Sampling Level		CPU Hours	Memory Usage (GB)	Number of Networks Found	Mean Num_Unsat_Hpol	Mean Saturation	Mean Score
∅		1.17	0.64	433	0	0.75	0.11
$\chi1$	Did Not Finish						
$\chi1\chi2$	Did Not Finish						

Monte Carlo HBNet

Chi Sampling Level	Monte Carlo Trajectories	CPU Hours	Memory Usage (GB)	Number of Networks Found	Mean Num_Unsat_Hpol	Mean Saturation	Mean Score
∅	1000	0.03	0.60	181	0	0.67	-0.88
∅	**10,000**	**0.03**	**0.61**	**389**	**0**	**0.68**	**-1.00**
∅	100,000	0.05	0.63	542	0	0.69	-0.61
$\chi1$	1000	0.08	0.84	357	0	0.86	-0.31
$\chi1$	**10,000**	**0.14**	**0.92**	**1616**	**0**	**0.91**	**0.43**
$\chi1$	100,000	0.52	1.22	5796	0	0.96	-0.20
$\chi1\chi2$	1000	0.22	1.51	436	0	0.84	0.38
$\chi1\chi2$	**10,000**	**0.34**	**1.63**	**2454**	**0**	**0.91**	**-0.72**
$\chi1\chi2$	100,000	1.44	2.29	12,835	0	0.94	-0.28

Table 2.S3. *Data for "Large Interface" case used to generate Figure 2.5 (MC HBNet data from Figure 2.5 is shown in bold). Mean values were calculated using the top 10 results reported by Rosetta*

HBNet

Chi Sampling Level		CPU Hours	Memory Usage (GB)	Number of Networks Found	Mean Num_Unsat_Hpol	Mean Saturation	Mean Score
∅	Did Not Finish						
$\chi1$	Did Not Finish						
$\chi1\chi2$	Did Not Finish						

Monte Carlo HBNet

Chi Sampling Level	Monte Carlo Trajectories	CPU Hours	Memory Usage (GB)	Number of Networks Found	Mean Num_Unsat_Hpol	Mean Saturation	Mean Score
∅	1000	0.02	0.38	386	0	0.73	3.78
∅	**10,000**	**0.06**	**0.40**	**2694**	**0**	**0.82**	**1.06**
∅	100,000	0.28	0.50	14232	0	0.82	1.07
∅	1,000,000	1.59	1.01	54317	0	0.83	0.73
$\chi1$	1000	0.03	0.46	645	0	0.77	2.17
$\chi1$	**10,000**	**0.12**	**0.50**	**5594**	**0**	**0.83**	**0.89**
$\chi1$	100,000	0.90	0.80	42,649	0	0.84	0.31
$\chi1$	1,000,000	13.49	3.01	277,032	0	0.88	1.52
$\chi1\chi2$	1000	0.06	0.79	683	0	0.75	1.50
$\chi1\chi2$	**10,000**	**0.16**	**0.83**	**6134**	**0**	**0.81**	**2.25**
$\chi1\chi2$	100,000	1.44	1.21	55,179	0	0.86	4.39
$\chi1\chi2$	1,000,000	30.34	4.58	439,059	0	0.90	2.00

Table 2.S4. *Data for "Small Helical Monomer" case used to generate Figure 2.5 (MC HBNet data from Figure 2.5 is shown in bold). Mean values were calculated using the top 10 results reported by Rosetta*

HBNet

Chi Sampling Level	CPU Hours	Memory Usage (GB)	Number of Networks Found
∅	0.01	0.53	0
$\chi1$	0.03	0.79	6
$\chi1\chi2$	44.81	1.45	23

Monte Carlo HBNet

Chi Sampling Level	Monte Carlo Trajectories	CPU Hours	Memory Usage (GB)	Number of Networks Found
∅	1000	0.01	0.39	1
∅	10,000	0.01	0.39	1
∅	**100,000**	**0.01**	**0.40**	**1**
∅	1,000,000	0.02	0.40	1
$\chi1$	1000	0.02	0.43	5
$\chi1$	10,000	0.03	0.44	14
$\chi1$	**100,000**	**0.04**	**0.54**	**22**
$\chi1$	1,000,000	0.08	0.98	27
$\chi1\chi2$	1000	0.04	0.50	8
$\chi1\chi2$	10,000	0.06	0.53	20
$\chi1\chi2$	**100,000**	**0.15**	**0.74**	**71**
$\chi1\chi2$	1,000,000	0.50	2.22	139

Table 2.S5. *Data for "Symmetric Homodimer 5J0K" case, used to generate Figure 2.6 (MC HBNet data from Figure 2.6 is shown in bold).*

HBNet

Chi Sampling Level	CPU Hours	Memory Usage (GB)	Number of Networks Found
∅	0.01	0.52	0
$\chi1$	0.03	0.79	9
$\chi1\chi2$	1.53	1.12	21

Monte Carlo HBNet

Chi Sampling Level	Monte Carlo Trajectories	CPU Hours	Memory Usage (GB)	Number of Networks Found
∅	1000	0.01	0.39	1
∅	10,000	0.01	0.40	1
∅	**100,000**	**0.02**	**0.41**	**1**
∅	1,000,000	0.02	0.41	1
$\chi1$	1000	0.02	0.43	10
$\chi1$	10,000	0.03	0.44	24
$\chi1$	**100,000**	**0.05**	**0.55**	**40**
$\chi1$	1,000,000	0.13	1.04	51
$\chi1\chi2$	1000	0.04	0.51	16
$\chi1\chi2$	10,000	0.07	0.51	42
$\chi1\chi2$	**100,000**	**0.16**	**0.68**	**207**
$\chi1\chi2$	1,000,000	0.51	1.77	438

Table 2.S6. *Data for "Symmetric Homodimer 5J10" case, used to generate Figure 2.6 (MC HBNet data from Figure 2.6 is shown in bold).*

HBNet

Chi Sampling Level	CPU Hours	Memory Usage (GB)	Number of Networks Found
∅	0.01	0.49	0
$\chi1$	0.04	0.79	2
$\chi1\chi2$	1.22	1.13	15

Monte Carlo HBNet

Chi Sampling Level	Monte Carlo Trajectories	CPU Hours	Memory Usage (GB)	Number of Networks Found
∅	1000	0.01	0.39	0
∅	10,000	0.01	0.39	0
∅	**100,000**	**0.01**	**0.39**	**0**
∅	1,000,000	0.01	0.39	0
$\chi1$	1000	0.02	0.44	5
$\chi1$	10,000	0.03	0.45	3
$\chi1$	**100,000**	**0.03**	**0.54**	**14**
$\chi1$	1,000,000	0.06	1.00	14
$\chi1\chi2$	1000	0.04	0.51	5
$\chi1\chi2$	10,000	0.06	0.52	14
$\chi1\chi2$	**100,000**	**0.14**	**0.72**	**53**
$\chi1\chi2$	1,000,000	0.38	2.07	121

Table 2.S7. *Data for "Symmetric Homodimer 5J0H" case, used to generate Figure 2.6 (MC HBNet data from Figure 2.6 is shown in bold).*

HBNet

Chi Sampling Level	CPU Hours	Memory Usage (GB)	Number of Networks Found
∅	0.01	0.52	2
$\chi1$	0.14	0.88	4
$\chi1\chi2$	16.28	1.34	28

Monte Carlo HBNet

Chi Sampling Level	Monte Carlo Trajectories	CPU Hours	Memory Usage (GB)	Number of Networks Found
∅	1000	0.01	0.40	0
∅	10,000	0.01	0.40	0
∅	**100,000**	**0.01**	**0.42**	**0**
∅	1,000,000	0.02	0.44	0
$\chi1$	1000	0.03	0.45	0
$\chi1$	10,000	0.05	0.47	1
$\chi1$	**100,000**	**0.09**	**0.61**	**2**
$\chi1$	1,000,000	0.21	1.46	3
$\chi1\chi2$	1000	0.05	0.53	0
$\chi1\chi2$	10,000	0.09	0.56	1
$\chi1\chi2$	**100,000**	**0.31**	**0.82**	**3**
$\chi1\chi2$	1,000,000	1.16	2.59	7

Table 2.S8. *Data for "Symmetric Homodimer 5IZS" case, used to generate Figure 2.6 (MC HBNet data from Figure 2.6 is shown in bold).*

ENDNOTE

[1]This work was previously published as Maguire, J. B., Boyken, S. E., Baker, D. & Kuhlman, B. Rapid Sampling of Hydrogen Bond Networks for Computational Protein Design. *J. Chem. Theory Comput.* (2018)

REFERENCES

1. Maguire, J. B., Boyken, S. E., Baker, D. & Kuhlman, B. Rapid Sampling of Hydrogen Bond Networks for Computational Protein Design. *J. Chem. Theory Comput.* (2018) doi:10.1021/acs.jctc.8b00033.

2. Judd, E. T., Stein, N., Pacheco, A. A. & Elliott, S. J. Hydrogen bonding networks tune proton-coupled redox steps during the enzymatic six-electron conversion of nitrite to ammonia. *Biochemistry* **53**, 5638–46 (2014).

3. Polander, B. C. & Barry, B. A. A hydrogen-bonding network plays a catalytic role in photosynthetic oxygen evolution. doi:10.1073/pnas.1200093109.

4. Sánchez-Azqueta, A. *et al.* A hydrogen bond network in the active site of Anabaena ferredoxin-NADP(+) reductase modulates its catalytic efficiency. *Biochim. Biophys. Acta* **1837**, 251–63 (2014).

5. Sigala, P. A. *et al.* Quantitative dissection of hydrogen bond-mediated proton transfer in the ketosteroid isomerase active site. *Proc. Natl. Acad. Sci. U. S. A.* **110**, (2013).

6. Joachimiak, L. A., Kortemme, T., Stoddard, B. L. & Baker, D. Computational Design of a New Hydrogen Bond Network and at Least a 300-fold Specificity Switch at a Protein−Protein Interface. *J. Mol. Biol.* **361**, 195–208 (2006).

7. Kuroda, D. & Gray, J. J. Shape complementarity and hydrogen bond preferences in protein-protein interfaces: Implications for antibody modeling and protein-protein docking. *Bioinformatics* btw197- (2016) doi:10.1093/bioinformatics/btw197.

8. Guo, H. & Salahub, D. R. Cooperative Hydrogen Bonding and Enzyme Catalysis. *Angew. Chem. Int. Ed. Engl.* **37**, 2985–2990 (1998).

9. Redzic, J. S. & Bowler, B. E. Role of hydrogen bond networks and dynamics in positive and negative cooperative stabilization of a protein. *Biochemistry* **44**, 2900–8 (2005).

10. Livesay, D. R., Huynh, D. H., Dallakyan, S. & Jacobs, D. J. Hydrogen bond networks determine emergent mechanical and thermodynamic properties across a protein family. *Chem. Cent. J.* **2**, 17 (2008).

11. Tatko, C. D., Nanda, V., Lear, J. D. & Degrado, W. F. Polar networks control oligomeric assembly in membranes. *J. Am. Chem. Soc.* **128**, 4170–1 (2006).

12. Lombardi, A. *et al.* Retrostructural analysis of metalloproteins: application to the design of a minimal model for diiron proteins. *Proc. Natl. Acad. Sci. U. S. A.* **97**, 6298–305 (2000).

13. Faiella, M. *et al.* An artificial di-iron oxo-protein with phenol oxidase activity. *Nat. Chem. Biol.* **5**, 882–4 (2009).

14. Reig, A. J. *et al*. Alteration of the oxygen-dependent reactivity of de novo Due Ferri proteins. *Nat. Chem.* **4**, 900–906 (2012).

15. Chino, M. *et al*. Artificial Diiron Enzymes with a De Novo Designed Four-Helix Bundle Structure. *Eur. J. Inorg. Chem.* **2015**, 3371–3390 (2015).

16. Zhang, S.-Q. *et al*. De Novo Design of Tetranuclear Transition Metal Clusters Stabilized by Hydrogen-Bonded Networks in Helical Bundles. *J. Am. Chem. Soc.* **140**, 1294–1304 (2018).

17. Guffy, S. L., Der, B. S. & Kuhlman, B. Probing the minimal determinants of zinc binding with computational protein design. *Protein Eng. Des. Sel.* **29**, 327–338 (2016).

18. O'Meara, M. J., Leaver-Fay, A. & Kuhlman, B. A Combined Covalent-Electrostatic Model of Hydrogen Bonding Improves Structure Prediction with Rosetta. *J Chem Theory Comput* **6**, 356–372 (2015).

19. Baker, E. N. & Hubbard, R. E. Hydrogen bonding in globular proteins. *Progress in Biophysics and Molecular Biology* vol. 44 97–179 (1984).

20. Stranges, P. B. & Kuhlman, B. A comparison of successful and failed protein interface designs highlights the challenges of designing buried hydrogen bonds. *Protein Sci.* **22**, 74–82 (2013).

21. Der, B. S. & Kuhlman, B. Strategies to control the binding mode of de novo designed protein interactions. *Curr. Opin. Struct. Biol.* **23**, 639–46 (2013).

22. Fleming, P. J. & Rose, G. D. Do all backbone polar groups in proteins form hydrogen bonds? *Protein Sci.* **14**, 1911–7 (2005).

23. Li, Z., Yang, Y., Zhan, J., Dai, L. & Zhou, Y. Energy functions in de novo protein design: current challenges and future prospects. *Annu. Rev. Biophys.* **42**, 315–35 (2013).

24. Boas, F. E. & Harbury, P. B. Potential energy functions for protein design. *Current Opinion in Structural Biology* vol. 17 199–204 (2007).

25. Alford, R. F. *et al*. The Rosetta All-Atom Energy Function for Macromolecular Modeling and Design. *J. Chem. Theory Comput.* **13**, 3031–3048 (2017).

26. Leaver-Fay, A., Kuhlman, B. & Snoeyink, J. An adaptive dynamic programming algorithm for the side chain placement problem. *Pac. Symp. Biocomput.* 16–27 (2005).

27. Boyken, S. *et al*. De novo design of protein homo-oligomers with modular hydrogen bond network-mediated specificty. *Science* **399**, 69–72 (2016).

28. Leaver-Fay, A. *et al*. Rosetta3: An Object-Oriented Software Suite for the Simulation and Design of Macromolecules. *Methods Enzymol.* **487**, 545–574 (2011).

29. Shapovalov, M. V. & Dunbrack, R. L. A smoothed backbone-dependent rotamer library for proteins derived from adaptive kernel density estimates and regressions. *Structure* **19**, 844–858 (2011).

30. Kortemme, T., Morozov, A. V. & Baker, D. An orientation-dependent hydrogen bonding potential improves prediction of specificity and structure for proteins and protein-protein complexes. *J. Mol. Biol.* **326**, 1239–1259 (2003).

31. Worth, C. L. & Blundell, T. L. Satisfaction of hydrogen-bonding potential influences the conservation of polar sidechains. *Proteins Struct. Funct. Bioinforma.* **75**, 413–429 (2009).

32. Rohl, C. A., Strauss, C. E. M., Misura, K. M. S. & Baker, D. Protein Structure Prediction Using Rosetta. *Methods Enzymol.* **383**, 66–93 (2004).

33. Durham, E., Dorr, B., Woetzel, N., Staritzbichler, R. & Meiler, J. Solvent accessible surface area approximations for rapid and accurate protein structure prediction. *J. Mol. Model.* **15**, 1093–108 (2009).

34. Rocklin, G. J. *et al.* Global analysis of protein folding using massively parallel design, synthesis, and testing. *Science (80-.).* **357**, 168–175 (2017).

35. Wang, G. & Dunbrack, R. L. PISCES: A protein sequence culling server. *Bioinformatics* **19**, 1589–1591 (2003).

36. Wang, G. & Dunbrack, R. L. PISCES: Recent improvements to a PDB sequence culling server. *Nucleic Acids Res.* **33**, (2005).

37. Smith, C. A. & Kortemme, T. Backrub-like backbone simulation recapitulates natural protein conformational variability and improves mutant side-chain prediction. *J. Mol. Biol.* **380**, 742–56 (2008).

CHAPTER 3: Optimizing Energy Landscape Perturbations for Improved Packing During Protein Design

3.1 Preface

This chapter is intentionally implementation-heavy and provides us with insight into sampling biases in Rosetta's all-atom (high-resolution) protein design protocols and possible solutions. The scope of these improvements impacts protein design as a whole, not just protein interface design, however we include interface design benchmarks in the process. In general, the goal of this chapter is to show that Rosetta's high-resolution design protocol can be improved by simple parameter-fitting and benchmarking these changes in a variety of protein design circumstances.

3.2 Background

FastRelax is Rosetta's default protocol for perturbing protein models to sample lower-energy conformations. FastRelax is a "mover"; a Rosetta protocol that changes the conformation of a protein.[1] FastRelax works by utilizing two simpler movers: PackRotamersMover and MinMover.[2–4]

PackRotamersMover generates a set of candidate sidechain conformations ("rotamers") for each residue position. Rotamers are randomly assigned to different positions repeatedly and the change is either accepted or rejected based on the Metropolis Criterion. PackRotamersMover can optionally perform design (i.e., change the amino acid identity at a position) by populating a position's rotamer set with rotamers of multiple amino acids. FastRelax does not enable this feature by default because the term "relax" implies maintaining a fixed sequence.

MinMover performs a gradient-based minimization in torsion-angle space for all atoms of the protein. MinMover cannot change amino acid identities and it generally does not make large-scale structural changes. A major benefit of MinMover is that it moves the backbone to accommodate changes in sidechain packing and to account for steric clashes.

FastRelax alternates between PackRotamersMover and MinMover four times, as shown in Figure 3.1. The first iteration decreases the score function's Lennard-Jones repulsive weight[5] to 2% of the original weight. Each subsequent iteration increases the repulsive weight until it is back to 100% in the fourth and final iteration. The decreased repulsive weight allows the protein to gradually resolve clashes as the structure becomes more refined.

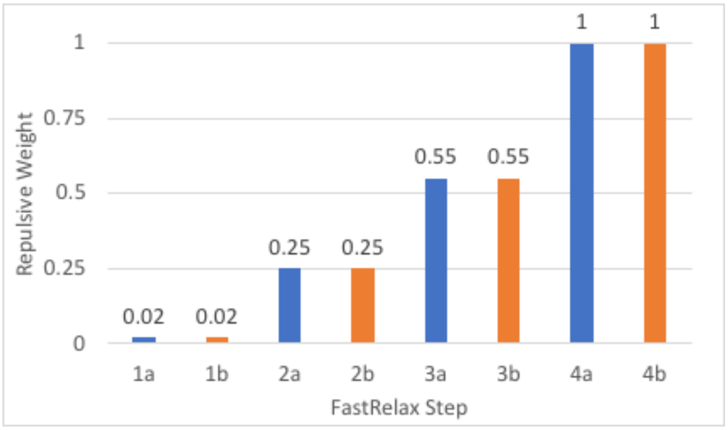


Figure 3.1 *FastRelax's default repulsive ramping scheme. PackRotamerMover steps are shown in blue and labeled "a", MinMover steps are shown in orange and labeled "b". Each step is annotated with the relative repulsive weight for that round, also shown on the y-axis, where a value of 1.0 is equal to the weight used for the Rosetta score function.*[5]

Protein engineers have had success using FastDesign, a derivative of FastRelax, to design proteins.[6–8] The sole difference between the two protocols is that FastDesign enables PackRotamersMover's ability to introduce mutations at user-defined positions, as mentioned above. Despite the success of FastDesign, careful guidance is required by the user to prevent FastDesign from inserting small hydrophobics into protein cores and reducing the total volume occupied by the protein. The goal of this project is to determine what is causing this undesired behavior and how to fix it.

3.3 Diagnosing the Problem

In order to evaluate how widespread the overdesign of small hydrophobic residues is, we set out to study the result of running FastDesign on dozens of native protein monomers. We used 52 crystal structures from the top8000 dataset[9] between 80-120 residues in length and with resolutions better than 1.5 Å. We ran FastDesign to redesign each protein's core 10 times and evaluated the results.

The first metric we measured was the radius of gyration ratio ("RG ratio") which divides the radius of gyration of the designed protein by that of the starting structure. A value less than 1 means that the designed protein is more compact that the native. We measured an average value of 0.97, which is consistent with the problematic behavior that we aim to correct.

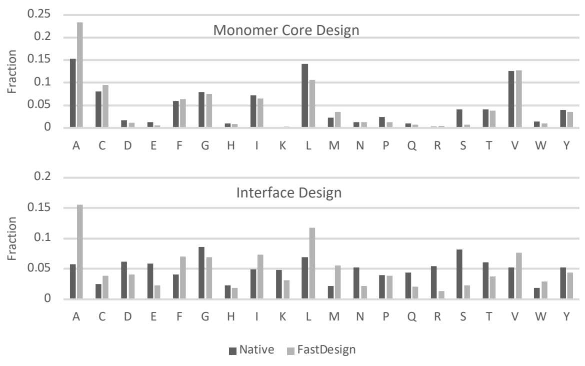


Figure 3.2 *Amino acid distribution of native proteins and proteins designed by FastDesign.*

We additionally measured the percentage of core residues that are designed to be alanine. Rosetta defines a residue as “core” if its C-alpha atom is within a minimum distance of 18 or more C-alpha atoms from other residues. On average, the native input structures have cores that are 14.7% alanine and FastDesign produces structures that have 28.4% alanine cores. Figure 3.2 compares the amino acid distributions of the native protein monomer cores and the protein cores after being designed by FastDesign, as well as the results of the same experiment performed on a set of native interfaces. These measurements support the hypothesis that FastDesign has a flaw that results in small core sidechains and the consequential shrinking in of the backbone.

To further track down the cause, we analyzed the structure of a protein as it progressed through the steps of FastDesign. We illustrate in Figure 3.3A how a native-like input structure behaves at each step of the FastDesign process. Step 1a mutates core positions to have larger sidechains, an expectable outcome of having only 2% repulsive weight. Despite the large sidechains in the core, the repulsive weight at step 1b is low enough to cause the chains of the protein to shrink in, towards one another. The repulsive weight is increased to 25% for steps 2a and 2b, causing some previously acceptable atomic distances to be now recognized as clashes. The backbone is unable to move for step 2a so step 1b’s backbone shrinking cannot be undone yet. Instead, Rosetta resolves the newly recognized clashes by mutating the core residue positions to have sidechains even smaller than their original identities. This is where the abundance of alanines is introduced. There are generally very few steric clashes remaining after step 2a because the sidechains are small, so the remaining steps do not alter the protein’s conformation dramatically. At the end of FastDesign, the backbone is still inwardly collapsed and the consequent alanine abundance is still present.

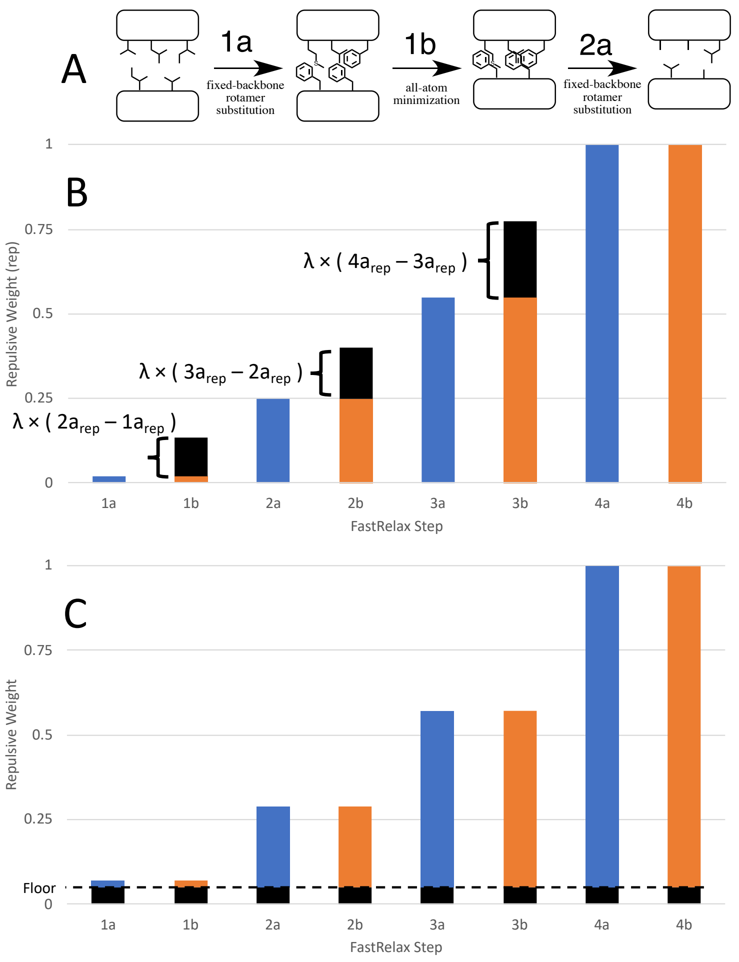


Figure 3.3 *Description of FastDesign's bias and the means to address it. (A) shows how FastDesign develops sampling error towards small sidechains. (B) and (C) illustrate the two hyperparameters that we plan to refit.*

3.4 Benefits of Increasing Repulsive Weight

The problem outlined in the previous section naively seemed to be solvable by increasing the repulsive energies in the first few rounds of FastDesign. Our plan was to try different repulsive ramping schemes to see if this problem goes away without sacrificing the Rosetta energies of FastDesign's final output.

Six of FastDesign's eight steps have non-standard repulsive weights for us to tune. Instead of sampling this six-dimensional space directly, we reduced it down to two dimensions: λ and *floor* (as shown in Figure 3.3B and 3.3C respectively). The *floor* dimension spans from 0 to 1 and pads each FastDesign step with a uniform weight increase. The remaining space above the floor is multiplied by *(1-floor)* to ensure that the ceiling is still 1. The λ dimension also spans 0 to 1 and only applies to the minimization steps of FastDesign. λ interpolates the repulsive weights between the neighboring packing steps, such that a larger value of λ results in a weight more similar to the subsequent packing step.

		Floor							
		0	0.02	0.04	0.05	0.06	0.08	0.09	0.1
	0	-3.61	-3.71	-3.75	-3.75	-3.75	-3.76	-3.76	-3.75
	0.05	-3.69	-3.75	-3.76	-3.76	-3.76	-3.75	-3.76	-3.75
λ	0.1	-3.71	-3.76	-3.76	-3.76	-3.76	-3.76	-3.75	-3.74
	0.15	-3.70	-3.76	-3.77	-3.76	-3.75	-3.75	-3.74	-3.75
	0.2	-3.69	-3.75	-3.76	-3.76	-3.76	-3.75	-3.74	-3.74

		Floor							
		0	0.02	0.04	0.05	0.06	0.08	0.09	0.1
	0	-3.57	-3.78	-3.89	-3.93	-3.95	-3.96	-3.96	-3.97
	0.05	-3.76	-3.88	-3.92	-3.95	-3.96	-3.97	-3.96	-3.95
λ	0.1	-3.82	-3.91	-3.93	-3.95	-3.97	-3.96	-3.97	-3.96
	0.15	-3.84	-3.91	-3.94	-3.96	-3.95	-3.95	-3.95	-3.95
	0.2	-3.82	-3.92	-3.93	-3.95	-3.94	-3.94	-3.95	-3.95

Figure 3.4 *Results of hyperparameter sampling for two of our design cases. Numbers shown in cells are in Rosetta Energy Units, where more negative is favorable. Top table is from RC Monomer Design, bottom is from RC Two-Sided Interface Design. As an example, the black box represents the parameter set to be used as MonomerDesign2019. The old default (legacy) is in the top left of each cell.*

We performed a coarse-grained grid search of these two dimensions on a variety of design cases, each explained in more detail in section 3.6.1. FastDesign ran on each case ten times for each structure (only five times each for interfaces due to their increased computational cost). Each case produced a heatmap of average Rosetta energies, two examples of which are shown in Figure 3.4. We identified the best set of parameters for each case and used that information to construct four relax scripts that cover all cases: InterfaceDesign2019, InterfaceRelax2019, MonomerDesign2019, and MonomerRelax2019, as shown in Table 3.1. The observed results for all cases are summarized in Table 3.2. The Rosetta energies improved for all design cases and some fixed sequence (“relax”) cases. The radius-of-gyration ratios and alanine percentages became closer to native as well.

Title	Floor	Lambda	1a	1b	2a	2b	3a	3b	4a	4b
InterfaceDesign2019	0.06	0.10	0.079	0.100	0.295	0.323	0.577	0.619	1	1
InterfaceRelax2019	0.05	0.05	0.069	0.080	0.288	0.302	0.573	0.594	1	1
MonomerDesign2019	0.04	0.15	0.059	0.092	0.280	0.323	0.568	0.633	1	1
MonomerRelax2019	0.02	0.05	0.040	0.051	0.265	0.280	0.559	0.581	1	1

***Table 3.1** Hyperparameters and repulsive weights for our four final protocols. PolarDesign2019 has the same weights as MonomerDesign2019. Repulsive weights for each FastDesign step are labeled using the convention set in Figure 3.1.*

		Energy Per Residue (REU)		Radius-Of-Gyration Ratio		Percent Alanine (Designable Positions)		
Set	**Case**	**Old**	**New**	**Old**	**New**	**Native**	**Old**	**New**
RC Monomer	Design (Core Positions)	-2.81	-2.96	0.97	1.00	14.7%	28.4%	13.9%
RC Monomer	Design	-3.61	-3.77	0.96	0.99	6.6%	16.2%	5.6%
RC Monomer	Relax	-2.78	-2.85	0.99	0.99			
Decoy Monomer #1	Relax	-2.70	-2.73	0.98	0.98			
Decoy Monomer #2	Relax	-2.39	-2.43	0.99	1.00			
Decoy Monomer #3	Design	-3.55	-3.63	1.00	1.04		13.1%	6.1%
RC Interface	Two-Sided Design	-3.57	-3.97	0.98	0.99	5.5%	17.6%	7.3%
RC Interface	One-Sided Design	-3.25	-3.57	0.99	1.00	5.8%	20.4%	7.3%
RC Interface	Relax (Interface Positions)	-3.06	-3.15	0.99	1.00			
Decoy Interface	One-Sided Design	-3.35	-3.68	0.99	0.99		18.8%	6.6%

***Table 3.2** Results of running new protocols on our diverse collection of design cases. For each case, we show the scores with the legacy (“Old”) protocol and the MonomerDesign2019/InterfaceDesign2019 (“New”) protocols. For design cases we include the percent of residues at designable positions that end up being alanine. Note, the native alanine percentage is omitted for computationally-generated backbones because they have no native sequence identity.*

3.5 Benefits of Ramping Reference Weight

Despite finding a lower Rosetta energy, the increased repulsive weights caused FastDesign to deviate even further from the native amino acid distribution, as shown in Figure 3.5. Alanine levels came closer to the native-like level, but large hydrophobic amino acids became over-sampled and polar/charged amino acids were under-sampled. Figure 3.5 shows how amino acids such as isoleucine (I), leucine (L), and tryptophan (W) became more abundant when sampling with the new repulsive weights. Conversely, polar amino acids such as glutamic acid (E), arginine (R), and serine (S) became less abundant.

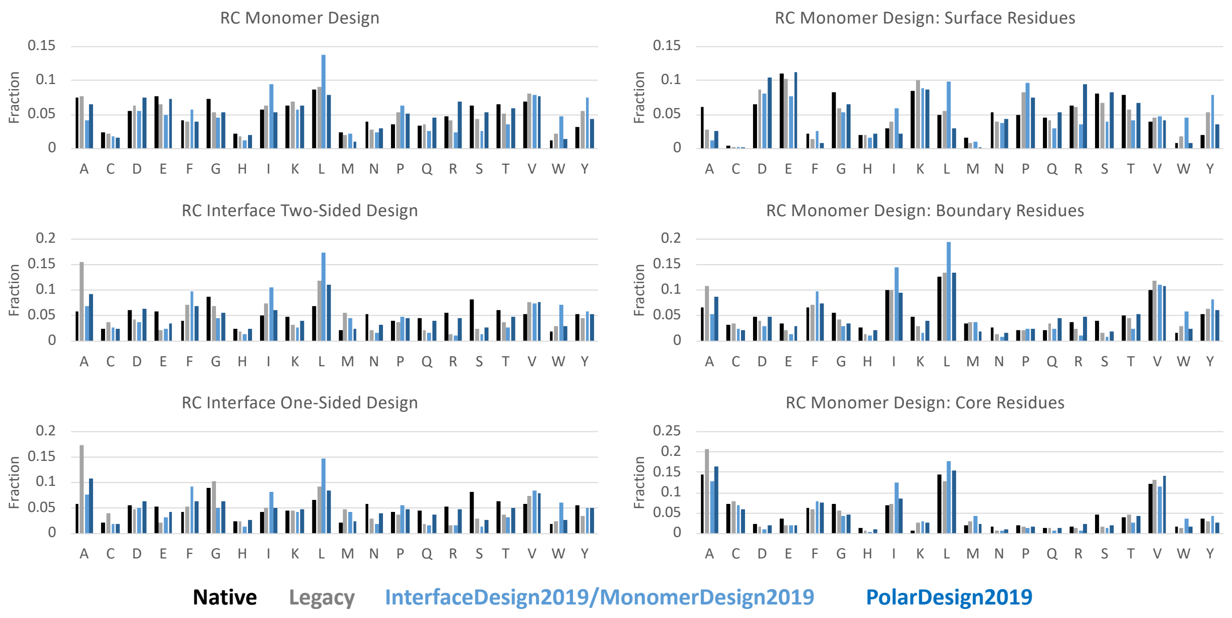


Figure 3.5 *Amino acid distributions for various benchmarks for native proteins and for designs of three different FastDesign protocols. InterfaceDesign2019 was used for the interface benchmarks and MonomerDesign2019 was used for the RC Monomer Design benchmark. The three columns on the right are a breakdown of the RC Monomer Design results by position in the protein. "Surface" residues are solvent exposed, "Core" residues are buried, and "Boundary" are partially exposed and partially buried.*

In order to counter this effect, we re-fit Rosetta's reference energies for each amino acid for the repulsive weight of each rotamer replacement step of FastDesign. Reference energies are fixed background energies for each amino acid intended to correct for unintended biases in the rest of the score function.[5] The PolarDesign2019 columns in Figure 3.5 show that the reference-energy-ramping technique generally improves the native-likeness of FastDesign's designs for both monomers and interfaces.

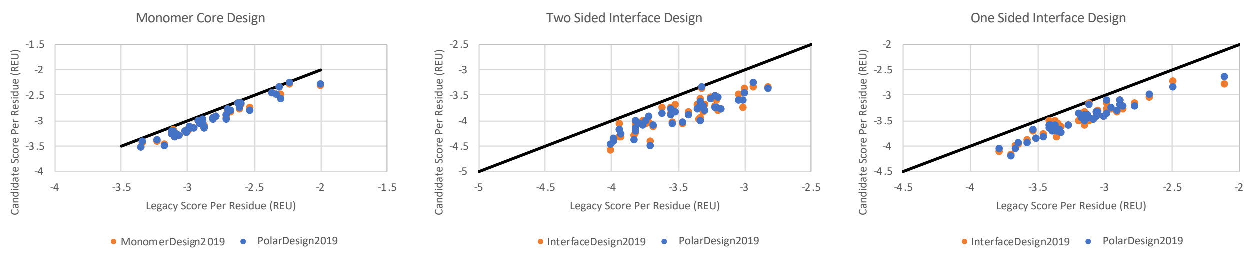


Figure 3.6 *Comparison of new design protocols with legacy for three design cases. Each point shows the average Rosetta score (normalized by residue count) that results from running FastDesign on a different protein. Both new protocol Rosetta scores are shown as a function of the score from the legacy protocols. The black diagonal line represents an equal score for the new protocol and legacy protocol. Points below the line represent protein structures that score better with the new protocol than the legacy protocol.*

In addition to the four relax scripts outlined in Table 3.1, we are also authoring PolarDesign2019. This script adds the aforementioned reference energy adjustments to MonomerDesign2019, which is the script that best serves as a one-size-fits-all solution for protein design. Figure 3.6 compares the Rosetta energies of designs created by PolarDesign2019 and MonomerDesign2019 against designs created with legacy FastDesign. PolarDesign2019 averages -0.41 REU/residue better than the legacy FastDesign in the case of two-sided interface design, while InterfaceDesign2019 averages -0.42 REU/residue of improvement. Both methods achieve comparable Rosetta energies, so it is reasonable to preliminarily conclude that PolarDesign2019 does not sacrifice quality in exchange for native-likeness.

3.6 Methods

3.6.1 Protein Structure Sets

We assembled a variety of sets of protein models, each either "RC" for relaxed crystal (see below) or "Decoy" for computer-generated models. *RC Monomer* is a set of relaxed crystal structures of monomers from the top8000 dataset[9] between 80-120 residues in length and with resolutions better than 1.5 Å. *RC Interface* is a set of relaxed crystal structures of interfaces dimeric crystal structures downloaded from the PDBbind database[10] with these filters: resolution must be better than 2.3 Å, the structures could not have more than two proteins chains and no ligand at the interface apart from HOH, SO4, CL, NA, MSE, and/or GOL. *Decoy Interface* structures were generated by using SEWING's AppendAssemblyMover[11] to design de novo 4-helical bundles that are designed to bind to the active form of the G protein, Gq alpha (see sections 3.8.1 and 3.8.2). *Decoy Monomer #1* was generated by running Rosetta's Abinitio demo without the final relax step. *Decoy Monomer #2* structures come from a previous experiment[1] in which Abinitio backbones were filtered based on downstream success with full-atom design. The goal of this set is to show that designs that previously performed well under legacy FastDesign do not get worse with our new variants. The *Decoy Monomer #3* set was created by stripping the Gq alpha chain from the *Decoy Interface* set, leaving only the protein chain designed by SEWING. Each set contains between 40 and 60 structures.

3.6.2 FastDesign Benchmarks

We had three different cases for the monomer sets: relax (fixed sequence), design (Rosetta was allowed to change the sequence of the protein), and core design (Rosetta was only allowed to change amino acid identities at core positions). The interface sets also had three cases: relax (fixed sequence), one-sided design (fixed sequence for one binding partner, the other binding partner could change sequence) and two-sided design (both chains can change sequences). We only ran protein sets that were logical for each case. For example, we did not run relax protocols on SEWING designs because there is no native amino acid identity for a SEWING-made structure.

3.6.3 Relaxing Crystal Structures

Each crystal structure was relaxed by running FastRelax 10 times and choosing the output structure with the lowest Rosetta energy. For this purpose, FastRelax is run with coordinate constraints (artificial energy bias that penalizes the CA atoms in the protein's backbone for deviating from their starting positions).

3.6.4 Reference Energy Fitting

For each repulsive weight of MonomerDesign2019, we ran optE_parallel (a Rosetta application). This application performs fixed-backbone rotamer substitution, allowing all residue positions to change amino acids. This process is repeated many times, each time modifying the amino acids' reference energies in an attempt to optimize for sequence recovery. optE_parallel was run three times for each repulsive weight and the outcome with the best score was used for PolarDesign2019.

3.7 Conclusion

We showed in this chapter that Rosetta's high-resolution design protocol had a sampling bias towards introducing small hydrophobic amino acids. We were able to correct this bias by defining and fitting two hyperparameters regarding FastDesign's repulsive weight ramping scheme. These corrections improved both sampling quality (resulting in lower Rosetta energies) and native-likeness of FastDesign's output. The native-likeness was further improved by re-parameterizing Rosetta's reference energies for each repulsive weight used by FastDesign, without apparent loss of design quality.

3.8 Supplemental Information

3.8.1 Script for SEWING Designs

```
<ROSETTASCRIPTS>
  <MOVERS>
    <AppendAssemblyMover name="aam"
model_file_name="inputs/smotifs_H_5_40_L_1_6_H_5_40.segments" partner_pdb="gaq.pdb"
hashed="false" required_resnums="43,44,45,46,47,48" minimum_cycles="10000"
maximum_cycles="11000" start_temperature="2" end_temperature="0.6"
pose_segment_starts="1,3,18,21,46,47" pose_segment_ends="2,17,20,45,46,60"
modifiable_terminus="C" output_partner="false" recover_lowest_assembly="true">
      <AssemblyScorers>
       <MotifScorer weight="1" />
       <InterModelMotifScorer weight="10" />
       <StartingNodeMotifScorer weight="1"/>
       <PartnerMotifScorer weight="1" />
      </AssemblyScorers>
      <AssemblyRequirements>
       <ClashRequirement clash_radius = "3.5" />
       <DsspSpecificLengthRequirement dssp_code="H" maximum_length="30"
minimum_length="10" />
       <DsspSpecificLengthRequirement dssp_code="L" maximum_length="4"
minimum_length="1" />
       <SizeInSegmentsRequirement maximum_size="7" minimum_size="1" />
      </AssemblyRequirements>
    </AppendAssemblyMover>
  </MOVERS>
  <PROTOCOLS>
    <Add mover="aam"/>
  </PROTOCOLS>
</ROSETTASCRIPTS>
```

3.8.2 Command Line Flags for SEWING Designs

```
-use_input_sc
-ignore_unrecognized_res
-linmem_ig 10
-nstruct 1000
-mh:match:aa1 false
-mh:match:aa2 false
-mh:score:use_ss1 true
-mh:score:use_ss2 true
-mh:path:motifs xsmax_bb_ss_AILV_resl0.8_msc0.3.rpm.bin.gz
-mh:path:scores_BB_BB xsmax_bb_ss_AILV_resl0.8_msc0.3
-mh:gen_reverse_motifs_on_load false
-mh::dump::max_rms 0.4
-pdb_comments true
-output_pose_energies_table false
-output_pose_cache_data false
-preserve_crystinfo true
```

REFERENCES

1. Tyka, M. D. *et al*. Alternate states of proteins revealed by detailed energy landscape mapping. *J Mol Biol* **405**, 607–618 (2011).

2. PackRotamersMover. https://www.rosettacommons.org/docs/latest/scripting_documentation/RosettaScripts/Movers/movers_pages/PackRotamersMover.

3. MinMover. https://www.rosettacommons.org/docs/latest/scripting_documentation/RosettaScripts/Movers/movers_pages/MinMover.

4. Kuhlman, B. *et al*. Design of a Novel Globular Protein Fold with Atomic-Level Accuracy. *Science (80-.)*. **302**, 1364–1368 (2003).

5. Alford, R. F. *et al*. The Rosetta All-Atom Energy Function for Macromolecular Modeling and Design. *J. Chem. Theory Comput.* **13**, 3031–3048 (2017).

6. Rocklin, G. J. *et al*. Global analysis of protein folding using massively parallel design, synthesis, and testing. *Science* **357**, 168–175 (2017).

7. Tyka, M. D., Jung, K. & Baker, D. Efficient sampling of protein conformational space using fast loop building and batch minimization on highly parallel computers. *J. Comput. Chem.* **33**, 2483–91 (2012).

8. Kaufmann, K. W., Lemmon, G. H., Deluca, S. L., Sheehan, J. H. & Meiler, J. Practically Useful: What the ROSETTA Protein Modeling Suite Can Do for You. doi:10.1021/bi902153g.

9. Keedy, D. A. *et al*. 8000 Filtered Structures. http://kinemage.biochem.duke.edu/databases/top8000.php (2012).

10. Liu, Z. *et al*. PDB-wide collection of binding data: Current status of the PDBbind database. *Bioinformatics* **31**, 405–412 (2015).

11. Guffy, S. L., Teets, F. D., Langlois, M. I. & Kuhlman, B. Protocols for Requirement-Driven Protein Design in the Rosetta Modeling Program. *J. Chem. Inf. Model.* **58**, 895–901 (2018).

CHAPTER 4: Benchmarking New Computational Techniques for Polar Protein-Protein Interface Design

4.1 Introduction

We established in Chapter 1 the premise that the Rosetta[1] protein modeling software is often unsuccessful at designing polar protein-protein interfaces. Stranges et al.[2] postulated that this failure is partially because Rosetta is unable to adequately sample hydrogen bonding partners for the polar atoms at the interface. We then spent Chapters 2 and 3 describing new computational techniques that might address this issue, namely Monte Carlo (MC) HBNet[3] and our new variants of FastDesign[4] including PolarDesign2019.

The goal of this chapter is to determine if these new methods actually improve Rosetta's ability to sample hydrogen bonds for polar interfaces. We will only perform computational benchmarks for the scope of this project. If the computational benchmarks pass, then we can pass this project on to experimental biochemists to see if these new methods perform well on real protein design projects.

The first of our two benchmarks is the "Average Trajectory Test". The goal of this test is to sample many variants of protein interface design protocols on a large number of structures to give us a better understanding of our landscape. Each protocol is run a small number of times on a large number of structures and the average result of each protocol is analyzed and compared to native interfaces.

Our other benchmark is the "Top Trajectory Test". The goal of this second test is to narrow in on a small number of protocols and test them extensively. This test aims to perform realistic, production-level interface design runs using a smaller suite of protocols than the first test. Because these tests are so computationally expensive, we will perform them on a smaller number of structures.

To generalize, the “Average Trajectory Test” tests broadly over many protocols (by varying the score function and sampling method), and many protein-protein interface structures. Conversely, the “Top Trajectory Test” tests narrowly but deeply on a small number of protein-protein interface structures. These tests measure Rosetta’s ability to design polar interfaces, ability to design well-packed interfaces, and ability to find hydrogen bonding partners for the polar atoms at the interface. Broadly speaking, these tests will be considered successful if MC HBNet and/or PolarDesign2019 are able to improve these abilities.

4.2 Average Trajectory Test

4.2.1 Methods

4.2.1.1 Structure Generation

The structures used for this benchmark are the same that comprised the *RC Interface* set in section 3.6.1.

4.2.1.2 Running Rosetta

Rosetta was run using RosettaScripts[5], an XML scripting interface to Rosetta protocols. The particular scripts used for this test are included in section 4.5.1. FastDesign.xml was used for all of the benchmarks except for the protocols that used HBNet, which used FastDesign.HBNet.xml. Rosetta was run 3 times for each structure for each protocol (‘-nstruct 3’ was passed via command line) and all 3 outputs were added to the pool of results.

RosettaScripts' xml files have runtime text replacement options denoted by double '%%' strings. These scripts had the following options: `%%script%%` was replaced with 'legacy'[4] or 'PolarDesign2019' and `%%sfxn%%` was substituted as 'score12',[6,7] 'talaris2013',[8] 'talaris2014',[9] 'ref2015' (Rosetta's current default),[10] or 'beta_nov16'. Note that most of these score functions may need additional command line flags to function correctly; it is best to consult documentation before attempting to use non-default score functions.

4.2.1.3 Evaluating Results

We have four metrics to analyze the results. First, we measure the fraction of interface residues on the variable-sequence (designable) side of the interface that are polar. These residues were classified as polar: DEHKNQRSTY. Interface residues were determined using Rosetta's InterGroupInterfaceByVector[11] protocol.

Second, we measure the packing quality of the interface. This was done by averaging the *packstat* and *shape complementarity*[12,13] values (both on a scale of 0 to 1 where 1 is higher quality) from Rosetta's Interface Analyzer.[2] We would normally use the score function energy itself to compare interface quality but it is meaningless to compare the energy from one score function with the energy from another.

Third, we measure the normalized number of unsatisfied polar atoms at the interface. For this metric, lower numbers are considered to be better. Note, this counts the number of polar atoms that have hydrogen bonding partners in the unbound state but not the bound state. If an atom has no hydrogen bonding partner in either state, it is not counted here. It is impossible in this system for atoms to only have hydrogen-bonding partners in the bound state because of implicit water interactions. We used Rosetta's BuriedUnsatHbonds[14] protocol for these calculations.

Fourth and finally, we also track the "per residue energy" of interface residues. This is another metric reported by Rosetta's Interface Analyzer and it essentially reports the mean Rosetta energy of the residues at the interface. More negative is considered better here, but that the scale and interpretation of this value is defined by the score function being used. It is meaningless to compare Rosetta scores from different score functions.

In addition to the four design-quality metrics, we also track the wall clock time of each protocol.

4.2.2 Results and Discussion

Our goal was to redesign native interfaces with various Rosetta protocols and score functions to determine the optimal way to design for hydrophilicity, hydrogen bond sampling, and overall packing quality. Specifically, we are performing one-sided design, a design condition in which one chain of the interface is fixed sequence while the other chain is allowed to make mutations. This is more challenging than two-sided design (where both chains can make mutations) because there will be polar residues on the fixed-sequence side that cannot be mutated away and Rosetta will need to find hydrogen-bonding partners for them.

We are testing all of the default score functions from *ref2015*[10] (the current default) back to *score12*[6,7] (the score function used to design the interfaces analyzed by Stranges et al.[2]), as well as *beta_nov16*, the heir apparent. We are also testing three different sampling protocols: legacy FastDesign, PolarDesign2019 FastDesign, and PolarDesign2019 FastDesign preceded by MC HBNet,[3] all of which were described in detail in Chapters 2 and 3. The metrics for the average design for each energy function/sampling method pair is shown in Table 4.1.

		Fraction Polar	Packing Quality	Unsatisfied Heavy Polar Atoms Per Interface Residue	Per Residue Energy (REU)	Runtime (Minutes)
	Native Interfaces	0.52	0.67	0.04		
Score Function	FastDesign protocol					
score12	legacy	0.51	0.67	0.10	-2.35	42
	PolarDesign2019	0.59	0.69	0.07	-2.46	41
talaris2013	legacy	0.45	0.68	0.07	-1.92	84
	PolarDesign2019	**0.51**	**0.69**	**0.05**	**-2.01**	**79**
	PolarDesign2019 (HBNet)	0.53	0.69	0.05	-1.96	66
talaris2014	legacy	0.43	0.68	0.08	-2.17	85
	PolarDesign2019	0.48	0.69	0.06	-2.31	77
ref2015	*legacy*	*0.30*	*0.66*	*0.08*	*-3.24*	*149*
	PolarDesign2019	0.43	0.69	0.05	-3.51	136
	PolarDesign2019 (HBNet)	**0.47**	**0.69**	**0.05**	**-3.38**	**107**
beta_nov16	legacy	0.33	0.65	0.06	-2.88	181
	PolarDesign2019	0.40	0.68	0.06	-3.22	174

Table 4.1 *Computational design quality metrics for native interface redesigns using different methods. Note that it is meaningless to compare the Per Residue Energy numbers across different score functions; we can only compare Per Residue Energy values between different FastDesign protocols within the same score function. The handpicked favorites are in bold and the current standard protocol is italicized. The first candidate (score12/legacy) is the protocol used to design the interfaces analyzed by Stranges et al.*[2] *All numbers are medians across all 373 interfaces. Score functions are listed in chronological order of development.*

The first entry of Table 4.1 confirms the pattern that Stranges et al. reported.[2] Legacy FastDesign using *score12* creates interfaces with native-like packing quality and native-like polar residue concentration, but it is poorly equipped to find hydrogen bonding partners for the polar atoms. Over twice as many polar atoms are unsatisfied in these designs than what we observe in native protein-protein interfaces.

The results in Table 4.1 show a trend towards hydrophobic design with the newer score functions. *score12* designs interfaces with 51% polar residues whereas *ref2015* drops that percentage to 30%. This can be remedied by using the PolarDesign2019 FastDesign protocol, which makes the interfaces more polar while also lowering the energy, increasing the packing quality, and decreasing the number of buried unsatisfied heavy atoms (unsats). Additionally, running MC HBNet prior to FastDesign further increases the number of polar residues without increasing the amount of buried unsats. MC HBNet does come with a mild decrease in projected stability, however. This is likely due to the fact that MC HBNet only optimizes for a small fraction of the final score terms when choosing mutations.

We handpicked two protocols that stood out as top performers and printed their results in bold in Table 4.1. Compared to the current standard (shown italicized), both of these protocols result in designs with improved polar residue densities, improved packing qualities, and improved levels of hydrogen bond satisfaction at the interface. In fact, all three of these metrics are near or better than the level observed at native interfaces.

Both of these protocols are also noticeably faster than the current standard. For stochastic protocols with large sample spaces such as interface design, one of the best ways to improve your results is to run the protocol more times.[3] We did not explicitly seek out this speed-up, but it does give potential users the sampling benefit of running more design trajectories without costing them more CPU-time.

When we compare the bold lines in Table 4.1 with the first entry (score12/legacy), we see that the new protocols are able to roughly match the properties of the score12/legacy entry but decrease the number of unsatisfied polar atoms at the interface by a factor of 2. This is a success within the scope of this benchmark because these new protocols directly address the concerns that Stranges et al.[2] raised about buried unsatisfied polar atoms without sacrificing other properties of the interface.

4.3 Top Trajectory Test

4.3.1 Methods

4.3.1.1 Structure Generation

The six structures used for this benchmark are a random subset of the *Decoy Interface* set in section 3.6.1. Each structure has one natively occurring protein chain and one *de novo* chain generated by SEWING.[15] For this benchmark, the native chain is restricted to its native sequence and the *de novo* chain is allowed to make unlimited mutations.

4.3.1.2 Running Rosetta

The Rosetta work in this section is nearly identical to the process described in section 4.2.1.2. One primary difference is that the Rosetta XML script is broken up into phases for this test (see section 4.5.2). All starting structures are preprocessed by running add_labels.xml, which decides which residues are allowed to be designed, and which residues are at the interface.

When benchmarking protocols with MC HBNet, the structures are then run with hbnet.xml. This script is run once per protocol (using command line flag '-nstruct 1') and will output up to 100 structures, each with a different hydrogen bond network.

All structures are then run with run.xml. When benchmarking protocols without MC HBNet, run.xml is executed 1000 times for each structure ('-nstruct 1000') resulting in 1000 designed structures for each of the six starting structures. Protocols with HBNet already have 100 states per benchmarking structure, so each of those states are executed with run.xml only 10 times ('-nstruct 10') to result in a total of 1000 structures.

We also introduced a third FastDesign variant: InterfaceDesign2019, an intermediate state between legacy and PolarDesign2019. InterfaceDesign2019 has the same repulsive weight ramping as PolarDesign2019 but does not have any reference weight ramping. Please see Chapter 3 for more details.

4.3.1.3 Evaluating Results

Each of the six starting interfaces will have 1000 candidate designs for each protocol. For each of the six interfaces, we picked the one design out of the 1000 that scored the best by Rosetta's score function ref2015[10], specifically the score per interface residue metric reported by the Interface Analyzer.[2] These best scoring structures were measured using three of the metrics from the "Average Trajectory Test": score per residue, interface polarity (fraction of interface residues that are classified as polar), and the number of unsatisfied heavy polar atoms at the interface. Please refer to section 4.2.1.3 for greater detail on these metrics.

4.3.2 Results and Discussion

The measurements in Table 4.1 represent the quality of an average trajectory, but we are also concerned with comparing the highest-quality trajectories for a given protocol. Computational protein designers are generally willing to run hundreds or thousands of trajectories to get only a handful of designs, so the average design trajectory does not need to be successful so long as the best designs are worthwhile.

We performed the "Top Trajectory Test" which ran FastDesign on a set of non-native interfaces (de novo binder to native target, labeled a-f in Table 4.2) 1000 times for each protocol to match the size of a realistic production run. The lowest scoring design from each set of trajectories is analyzed in Table 4.2. For this test, we performed all runs using the ref2015 score function. Reversion to talaris2013 had promising results in the previous test but we decided to stick with ref2015 for this more in-depth look as it is the current default score function.

As mentioned in Chapter 3, there are two differences between the legacy and PolarDesign2019 protocols: repulsive weight ramping and reference weight ramping. For this test, we are including an intermediary named InterfaceDesign2019. The only difference between legacy and InterfaceDesign2019 is the repulsive weight ramping and the only difference between InterfaceDesign2019 and PolarDesign2019 is the reference weight ramping.

		Score Per Interface Residue (REU)	Interface Polarity	Unsatisfied Heavy Polar Atoms Per Interface Residue
A	legacy	-3.21	0.16	0.11
	InterfaceDesign2019	-3.48	0.28	0.12
	PolarDesign2019	-3.44	0.28	0.07
	PolarDesign2019 (HBNet)	**-3.35**	**0.59**	**0.06**
b	legacy	-3.29	0.41	0.08
	InterfaceDesign2019	-3.57	0.30	0.11
	PolarDesign2019	-3.62	0.54	0.11
	PolarDesign2019 (HBNet)	**-3.59**	**0.57**	**0.05**
c	legacy	-3.13	0.27	0.07
	InterfaceDesign2019	-3.73	0.30	0.06
	PolarDesign2019	**-3.48**	**0.49**	**0.07**
	PolarDesign2019 (HBNet)	-3.37	0.43	0.10
d	legacy	-3.40	0.27	0.11
	InterfaceDesign2019	-3.68	0.30	0.06
	PolarDesign2019	**-3.67**	**0.57**	**0.06**
	PolarDesign2019 (HBNet)	-3.58	0.41	0.07
e	legacy	-3.34	0.15	0.06
	InterfaceDesign2019	-3.61	0.23	0.07
	PolarDesign2019	**-3.62**	**0.46**	**0.05**
	PolarDesign2019 (HBNet)	**-3.56**	**0.54**	**0.05**
f	legacy	-3.51	0.22	0.08
	InterfaceDesign2019	-4.05	0.35	0.05
	PolarDesign2019	**-4.00**	**0.43**	**0.06**
	MCHBNet PolarDesign2019	**-4.03**	**0.57**	**0.08**

Table 4.2 *Results of "Top Trajectory Test". Current standard protocol is named "legacy", handpicked favorites for each case are shown in bold.*

Using non-native interfaces gives us the benefit of distinguishing between poor sampling and under-sampling. A protocol that did *absolutely nothing* to the protein would look good in Table 4.1 because the output of the native protein would be very native-like. The native interfaces already have native-like qualities so a null-operation would output native-like designs. That same protocol would be exposed as weak in this next test because these non-native interfaces do not have a stable starting point. Rosetta needs to be able to sample well in order to be impressive here.

Recall that native interfaces have 52% polar residues on average. Our goal was to sample designs that had close to that concentration without sacrificing stability. Based on the combination of score and interface polarity, the legacy and InterfaceDesign2019 protocols did not perform as well as PolarDesign2019. PolarDesign2019 without MC HBNet was able to design the hand-picked favorite interfaces in cases ‘c’ and ‘d’ but benefited from being preceded by MC HBNet in cases ‘a’ and ‘b’.

In case ‘a’, MC HBNet was able to boost interface polarity from 28% to 59% at a cost less than 0.1 REU/residue. PolarDesign2019 designs were polar enough alone in case ‘b’, but MC HBNet was able to decrease the number of polar unsatisfied atoms at the interface by a factor of two. We deemed ‘e’ and ‘f’ to be toss-ups with regards to MC HBNet.

4.4 Conclusion

For the scope of this project, these benchmarks show success. Stranges et al. originally reported that Rosetta introduces too many unsatisfied polar atoms when making polar interfaces, such that only Rosetta’s hydrophobic interface designs were consistently stable.[2] Our new FastDesign variants with the optional help of MC HBNet and the utilization of a modern score function were able to design polar interfaces with half of the number of unsatisfied polar atoms compared to the technique Stranges et al. audited, all without sacrificing interface packing quality or polarity (Table 4.1, compare the score12/legacy line to the lines in bold).

When we took a deeper look at production runs using the modern default score function, our new protocols were able to design interfaces with native-like interface polarity while resulting in *fewer* unsatisfied polar atoms than even Rosetta's more hydrophobic interface designs (and a more favorable Rosetta score to boot). This shows us that we have improved Rosetta's ability to sample hydrogen bonding partners and thus more promising polar interface designs.

Albeit, there is a limit to amount that we can learn from these computational benchmarks. Success within the Rosetta score function does not always result in success in the test tube. Stronger conclusions will be made when PolarDesign2019 and MC HBNet are used for the design of interfaces that undergo experimental testing.

4.5 Supplemental Information

4.5.1 Rosetta Scripts for "Average Trajectory Test"

FastDesign.xml

```
<ROSETTASCRIPTS>

  <RESIDUE_SELECTORS>
    <ResiduePDBInfoHasLabel name="interface" property="INTERFACE" />
    <ResiduePDBInfoHasLabel name="design"    property="DESIGN" />
    <ResiduePDBInfoHasLabel name="repack"    property="REPACK" />
    <ResiduePDBInfoHasLabel name="fixed"     property="FIXED" />
  </RESIDUE_SELECTORS>

  <TASKOPERATIONS>
    <ExtraRotamersGeneric name="extra_chi" ex1="1" ex2="1" />
    <IncludeCurrent name="incl_curr" />
    <SetIGType name="lin<ROSETTASCRIPTS>

  <RESIDUE_SELECTORS>
    <Layer name="core_res" select_core="1" select_boundary="0" select_surface="0" />
    <Not name="not_core_res" selector="core_res"/>
    <StoredResidueSubset name="original_core" subset_name="core" />

    <Chain name="chain1" chains="1"/>
    <Chain name="chain2" chains="2"/>
    <InterfaceByVector             name="interface"             grp1_selector="chain1"
grp2_selector="chain2"/>
    <StoredResidueSubset name="original_interface" subset_name="intfc" />
    <Not name="not_interface" selector="original_interface"/>
    <Not name="two_sided_design" selector="original_interface"/>

    <Or name="one_sided_design" selectors="not_interface,chain2"/>

    <Or name="relax_only" selectors="chain1,chain2"/>

    <Not name="designable" selector="%%case%%"/>

    <ResiduePDBInfoHasLabel name="hbnet" property="HBNet"/>
  </RESIDUE_SELECTORS>

  <TASKOPERATIONS>
    <DisallowIfNonnative name="no_big_polars" disallow_aas="RKHNQDE"/>

    <DisallowIfNonnative name="no_polars" disallow_aas="RKHNQDESTY"/>

    <IncludeCurrent name="keep_curr"/>

    <ExtraRotamersGeneric name="extrachi"
                      ex1="1" ex2="1" ex3="0" ex4="0"
                      ex1_sample_level="1"  ex2_sample_level="1"  ex3_sample_level="0"
ex4_sample_level="0"
                      extrachi_cutoff="18"/>

    <OperateOnResidueSubset name="repack_non_interface" selector="one_sided_design">
      <RestrictToRepackingRLT/>
    </OperateOnResidueSubset>

    <OperateOnResidueSubset name="fix_non_interface" selector="not_interface">
      <PreventRepackingRLT/>
    </OperateOnResidueSubset>

    <SetIGType name="linmem_ig" lin_mem_ig="true"/>

    <OperateOnResidueSubset name="fix_hbnet" selector="hbnet">
      <PreventRepackingRLT/>
    </OperateOnResidueSubset>

  </TASKOPERATIONS>

  <SCOREFXNS>
    <ScoreFunction name="sfxn" weights="%%sfxn%%"/>
  </SCOREFXNS>

  <SIMPLE_METRICS>
    <TimingProfileMetric name="timing" />
    <SelectedResidueCountMetric name="interface_size" residue_selector="interface" />
    <SequenceMetric name="seq" residue_selector="original_interface" />
    <SequenceMetric name="des_seq" residue_selector="designable" />
  </SIMPLE_METRICS>

  <FILTERS>
    <BuriedUnsatHbonds     name="buh_sc_heavy"     report_sc_heavy_atom_unsats="true"
cutoff="99999" residue_selector="interface"/>
    <BuriedUnsatHbonds     name="buh_bb_heavy"     report_bb_heavy_atom_unsats="true"
cutoff="99999" residue_selector="interface"/>
    <BuriedUnsatHbonds   name="buh_H"   report_nonheavy_unsats="true"   cutoff="99999"
residue_selector="interface"/>

    <ResidueCount      name="run_num_polars_des"      include_property="POLAR,CHARGED"
residue_selector="designable" />
    <ResidueCount name="num_designable" residue_selector="designable" />

    <ReadPoseExtraScoreFilter     name="read_preNumPolar"     term_name="preNumPolar"
threshold="99999"/>
    <ReadPoseExtraScoreFilter    name="read_postNumPolar"    term_name="postNumPolar"
threshold="99999"/>

    <CalculatorFilter name="change_in_polar_count" equation="A - B" threshold="99999"
>
      <Var name="A" filter="read_postNumPolar"/>
      <Var name="B" filter="read_preNumPolar"/>
    </CalculatorFilter>

    <CalculatorFilter name="percent_change_in_polar_count" equation="( A - B ) / C"
threshold="99999" >
      <Var name="A" filter="read_postNumPolar"/>
      <Var name="B" filter="read_preNumPolar"/>
      <Var name="C" filter="num_designable"/>
    </CalculatorFilter>

  </FILTERS>

  <MOVERS>
    <StoreResidueSubset             name="store_core"             subset_name="core"
residue_selector="core_res" overwrite="1" />
    <StoreResidueSubset          name="store_interface"          subset_name="intfc"
residue_selector="interface" overwrite="1" />

    <VirtualRoot name="vr" />
    <AddConstraintsToCurrentConformationMover name="cc" bound_width="0" CA_only="1" />

    <FastDesign name="RelaxDesign" repeats="5" disable_design="false" scorefxn="sfxn"
task_operations="keep_curr,repack_non_interface,extrachi,linmem_ig,fix_non_interface"
relaxscript="%%script%%"/>
    <InterfaceAnalyzerMover    name="IfaceAnalyzer"    scorefxn="sfxn"    packstat="1"
interface_sc="1" pack_input="0" pack_separated="1" jump="1" tracer="false" />

    <FilterReportAsPoseExtraScoresMover                              name="preBUNS1"
report_as="BUNS_sc_heavy_before" filter_name="buh_sc_heavy"/>
    <FilterReportAsPoseExtraScoresMover                              name="preBUNS2"
report_as="BUNS_bb_heavy_before" filter_name="buh_bb_heavy"/>
```

```
    <FilterReportAsPoseExtraScoresMover   name="preBUNS3"   report_as="BUNS_H_before"
filter_name="buh_H"/>

    <FilterReportAsPoseExtraScoresMover  name="BUNS1"  report_as="BUNS_sc_heavy_after"
filter_name="buh_sc_heavy"/>
    <FilterReportAsPoseExtraScoresMover  name="BUNS2"  report_as="BUNS_bb_heavy_after"
filter_name="buh_bb_heavy"/>
    <FilterReportAsPoseExtraScoresMover     name="BUNS3"     report_as="BUNS_H_after"
filter_name="buh_H"/>

    <FilterReportAsPoseExtraScoresMover   name="preNumPolar"   report_as="preNumPolar"
filter_name="run_num_polars_des"/>
    <FilterReportAsPoseExtraScoresMover  name="postNumPolar"  report_as="postNumPolar"
filter_name="run_num_polars_des"/>

    <FilterReportAsPoseExtraScoresMover                name="CalcChangeInPolarCount"
report_as="dNumPolar" filter_name="change_in_polar_count"/>
    <FilterReportAsPoseExtraScoresMover                 name="CalcChangeInPolarFrac"
report_as="dFracPolar" filter_name="percent_change_in_polar_count"/>

    <RunSimpleMetrics name="t1" metrics="timing" prefix="t1_" />
    <RunSimpleMetrics name="t2" metrics="timing" prefix="t2_" />
    <RunSimpleMetrics name="rsm" metrics="interface_size" prefix="int_size_" />

    <RunSimpleMetrics name="seq1" metrics="seq" prefix="int_seq_before" />
    <RunSimpleMetrics name="seq1a" metrics="des_seq" prefix="des_seq_before" />
    RunSimpleMetrics              name="npol1"              metrics="num_polars_des"
prefix="num_polars_des_before"

    <RunSimpleMetrics name="seq2" metrics="seq" prefix="int_seq_after" />
    <RunSimpleMetrics name="seq2a" metrics="des_seq" prefix="des_seq_after" />
    RunSimpleMetrics              name="npol2"              metrics="num_polars_des"
prefix="num_polars_des_after"
  </MOVERS>

  <PROTOCOLS>
    Add mover="vr"
    Add mover="cc"

    <Add mover="store_core"/>
    <Add mover="store_interface"/>
    <Add mover="rsm"/>
    <Add mover="seq1"/>
    <Add mover="seq1a"/>
    <Add mover="preNumPolar"/>

    <Add mover="preBUNS1"/>
    <Add mover="preBUNS2"/>
    <Add mover="preBUNS3"/>

    <Add mover="t1"/>
    <Add mover="RelaxDesign"/>
    <Add mover="t2"/>
    <Add mover="IfaceAnalyzer"/>
    <Add mover="seq2"/>
    <Add mover="seq2a"/>
    <Add mover="postNumPolar"/>

    <Add mover="BUNS1"/>
    <Add mover="BUNS2"/>
    <Add mover="BUNS3"/>

    <Add mover="CalcChangeInPolarFrac"/>
    <Add mover="CalcChangeInPolarCount"/>
  </PROTOCOLS>

  <OUTPUT scorefxn="sfxn"/>
</ROSETTASCRIPTS>
```

FastDesign.HBNet.xml

```
<ROSETTASCRIPTS>

  <RESIDUE_SELECTORS>
    <Layer name="core_res" select_core="1" select_boundary="0" select_surface="0" />
    <Not name="not_core_res" selector="core_res"/>

    <StoredResidueSubset name="original_core" subset_name="core" />

    <Chain name="chain1" chains="1"/>
    <Chain name="chain2" chains="2"/>
    <InterfaceByVector            name="interface"            grp1_selector="chain1"
grp2_selector="chain2"/>
    <StoredResidueSubset name="original_interface" subset_name="intfc" />
    <Not name="not_interface" selector="original_interface"/>
    <Not name="two_sided_design" selector="original_interface"/>

    <Or name="one_sided_design" selectors="not_interface,chain2"/>

    <Or name="relax_only" selectors="chain1,chain2"/>

    <Not name="designable" selector="%%case%%"/>

    <ResiduePDBInfoHasLabel name="hbnet" property="HBNet"/>
  </RESIDUE_SELECTORS>

  <TASKOPERATIONS>
    <DisallowIfNonnative name="no_big_polars" disallow_aas="RKHNQDE"/>

    <DisallowIfNonnative name="no_polars" disallow_aas="RKHNQDESTY"/>

    <IncludeCurrent name="keep_curr"/>

    <ExtraRotamersGeneric name="extrachi" ex1="1" ex2="1" ex3="0" ex4="0"
                  ex1_sample_level="1"  ex2_sample_level="1"  ex3_sample_level="0"
ex4_sample_level="0" extrachi_cutoff="18"/>

    <OperateOnResidueSubset name="repack_non_interface" selector="one_sided_design">
      <RestrictToRepackingRLT/>
    </OperateOnResidueSubset>

    <OperateOnResidueSubset name="fix_non_interface" selector="not_interface">
      <PreventRepackingRLT/>
    </OperateOnResidueSubset>

    <SetIGType name="linmem_ig" lin_mem_ig="true"/>

    <OperateOnResidueSubset name="fix_hbnet" selector="hbnet">
      <PreventRepackingRLT/>
    </OperateOnResidueSubset>

  </TASKOPERATIONS>
  <SCOREFXNS>
    <ScoreFunction name="sfxn" weights="%%sfxn%%"/>
  </SCOREFXNS>

  <SIMPLE_METRICS>
    <TimingProfileMetric name="timing" />
    <SelectedResidueCountMetric name="interface_size" residue_selector="interface" />
    <SequenceMetric name="seq" residue_selector="original_interface" />
    <SequenceMetric name="des_seq" residue_selector="designable" />
  </SIMPLE_METRICS>

  <FILTERS>
    <BuriedUnsatHbonds     name="buh_sc_heavy"     report_sc_heavy_atom_unsats="true"
cutoff="99999" residue_selector="interface"/>
    <BuriedUnsatHbonds     name="buh_bb_heavy"     report_bb_heavy_atom_unsats="true"
cutoff="99999" residue_selector="interface"/>
    <BuriedUnsatHbonds   name="buh_H"   report_nonheavy_unsats="true"   cutoff="99999"
residue_selector="interface"/>

    <ResidueCount      name="run_num_polars_des"      include_property="POLAR,CHARGED"
residue_selector="designable" />
    <ResidueCount name="num_designable" residue_selector="designable" />

    <ReadPoseExtraScoreFilter     name="read_preNumPolar"     term_name="preNumPolar"
threshold="99999"/>
    <ReadPoseExtraScoreFilter    name="read_postNumPolar"    term_name="postNumPolar"
threshold="99999"/>

    <CalculatorFilter name="change_in_polar_count" equation="A - B" threshold="99999"
>
      <Var name="A" filter="read_postNumPolar"/>
      <Var name="B" filter="read_preNumPolar"/>
    </CalculatorFilter>

    <CalculatorFilter name="percent_change_in_polar_count" equation="( A - B ) / C"
threshold="99999" >
      <Var name="A" filter="read_postNumPolar"/>
      <Var name="B" filter="read_preNumPolar"/>
      <Var name="C" filter="num_designable"/>
    </CalculatorFilter>

  </FILTERS>

  <MOVERS>
    <StoreResidueSubset              name="store_core"              subset_name="core"
residue_selector="core_res" overwrite="1" />
    <StoreResidueSubset           name="store_interface"           subset_name="intfc"
residue_selector="interface" overwrite="1" />

    <VirtualRoot name="vr" />
    <AddConstraintsToCurrentConformationMover name="cc" bound_width="0" CA_only="1" />

    <FastDesign name="RelaxDesign" repeats="5" disable_design="false" scorefxn="sfxn"
task_operations="keep_curr,repack_non_interface,extrachi,linmem_ig,fix_non_interface,
fix_hbnet" relaxscript="%%script%% "/>
```

```
    <HBNetStapleInterface   hb_threshold="-0.65"   store_network_scores_in_pose="true"
secondary_threshold="-0.5"       write_cst_files="false"       max_network_size="100"
max_unsat_Hpol="3"       design_residues="STKHYWNQDE"             monte_carlo="true"
total_num_mc_runs="100000"
task_operations="keep_curr,repack_non_interface,extrachi,fix_non_interface"
scorefxn="sfxn"  name="HBNet"  max_networks_per_pose="10"  min_networks_per_pose="1"
allow_no_hbnets="true"/>


    <InterfaceAnalyzerMover    name="IfaceAnalyzer"    scorefxn="sfxn"    packstat="1"
interface_sc="1" pack_input="0" pack_separated="1" jump="1" tracer="false" />

    <FilterReportAsPoseExtraScoresMover                              name="preBUNS1"
report_as="BUNS_sc_heavy_before" filter_name="buh_sc_heavy"/>
    <FilterReportAsPoseExtraScoresMover                              name="preBUNS2"
report_as="BUNS_bb_heavy_before" filter_name="buh_bb_heavy"/>
    <FilterReportAsPoseExtraScoresMover   name="preBUNS3"   report_as="BUNS_H_before"
filter_name="buh_H"/>

    <FilterReportAsPoseExtraScoresMover  name="BUNS1"  report_as="BUNS_sc_heavy_after"
filter_name="buh_sc_heavy"/>
    <FilterReportAsPoseExtraScoresMover  name="BUNS2"  report_as="BUNS_bb_heavy_after"
filter_name="buh_bb_heavy"/>
    <FilterReportAsPoseExtraScoresMover     name="BUNS3"     report_as="BUNS_H_after"
filter_name="buh_H"/>

    <FilterReportAsPoseExtraScoresMover   name="preNumPolar"   report_as="preNumPolar"
filter_name="run_num_polars_des"/>
    <FilterReportAsPoseExtraScoresMover  name="postNumPolar"  report_as="postNumPolar"
filter_name="run_num_polars_des"/>

    <FilterReportAsPoseExtraScoresMover                name="CalcChangeInPolarCount"
report_as="dNumPolar" filter_name="change_in_polar_count"/>
    <FilterReportAsPoseExtraScoresMover                 name="CalcChangeInPolarFrac"
report_as="dFracPolar" filter_name="percent_change_in_polar_count"/>

    <RunSimpleMetrics name="t1" metrics="timing" prefix="t1_" />
    <RunSimpleMetrics name="t2" metrics="timing" prefix="t2_" />
    <RunSimpleMetrics name="rsm" metrics="interface_size" prefix="int_size_" />

    <RunSimpleMetrics name="seq1" metrics="seq" prefix="int_seq_before" />
    <RunSimpleMetrics name="seq1a" metrics="des_seq" prefix="des_seq_before" />
    RunSimpleMetrics              name="npol1"              metrics="num_polars_des"
prefix="num_polars_des_before"

    <RunSimpleMetrics name="seq2" metrics="seq" prefix="int_seq_after" />
    <RunSimpleMetrics name="seq2a" metrics="des_seq" prefix="des_seq_after" />
    RunSimpleMetrics              name="npol2"              metrics="num_polars_des"
prefix="num_polars_des_after"
  </MOVERS>

  <PROTOCOLS>
    Add mover="vr"
    Add mover="cc"
    <Add mover="store_core"/>
    <Add mover="store_interface"/>
    <Add mover="rsm"/>
    <Add mover="seq1"/>
    <Add mover="seq1a"/>
    <Add mover="preNumPolar"/>

    <Add mover="preBUNS1"/>
    <Add mover="preBUNS2"/>
    <Add mover="preBUNS3"/>

    <Add mover="t1"/>
    <Add mover="HBNet"/>
    <Add mover="RelaxDesign"/>
    <Add mover="t2"/>
    <Add mover="IfaceAnalyzer"/>

    <Add mover="seq2"/>
    <Add mover="seq2a"/>
    <Add mover="postNumPolar"/>

    <Add mover="BUNS1"/>
    <Add mover="BUNS2"/>
    <Add mover="BUNS3"/>

    <Add mover="CalcChangeInPolarFrac"/>
    <Add mover="CalcChangeInPolarCount"/>
  </PROTOCOLS>

  <OUTPUT scorefxn="sfxn"/>
</ROSETTASCRIPTS>
```

4.5.2 Rosetta Scripts for “Top Trajectory Test”

add_labels.xml:

```
<ROSETTASCRIPTS>

  <RESIDUE_SELECTORS>
    <Chain             name="chain1" chains="1"/>
    <Chain             name="chain2" chains="2"/>
    <InterfaceByVector            name="interface"            grp1_selector="chain1"
grp2_selector="chain2"/>
    <And               name="design" selectors="interface,chain1"/>
    <And               name="repack" selectors="interface,chain2"/>
    <Not               name="fixed"  selector="interface"/>
  </RESIDUE_SELECTORS>

  <MOVERS>
    <AddResidueLabel       name="interface_label"       residue_selector="interface"
label="INTERFACE"/>
    <AddResidueLabel    name="design_label"                 residue_selector="design"
label="DESIGN"/>
    <AddResidueLabel    name="repack_label"                 residue_selector="repack"
label="REPACK"/>
    <AddResidueLabel    name="fixed_label"                   residue_selector="fixed"
label="FIXED"/>
  </MOVERS>

  <PROTOCOLS>
    <Add mover="interface_label"/>
    <Add mover="design_label"/>
    <Add mover="repack_label"/>
    <Add mover="fixed_label"/>
  </PROTOCOLS>

</ROSETTASCRIPTS>
```

hbnet.xml:

```
<ROSETTASCRIPTS>

  <RESIDUE_SELECTORS>
    <ResiduePDBInfoHasLabel name="interface" property="INTERFACE" />
    <ResiduePDBInfoHasLabel name="design"    property="DESIGN" />
    <ResiduePDBInfoHasLabel name="repack"    property="REPACK" />
    <ResiduePDBInfoHasLabel name="fixed"     property="FIXED" />
  </RESIDUE_SELECTORS>

  <TASKOPERATIONS>
    <ExtraRotamersGeneric name="extra_chi" ex1="1" ex2="1" />
    <IncludeCurrent name="incl_curr" />
    <SetIGType name="linmem_ig" lin_mem_ig="true" /> <!-- -linmem_ig 10 -->

    <OperateOnResidueSubset name="fix" selector="fixed" >
      <PreventRepackingRLT/>
    </OperateOnResidueSubset>

    <OperateOnResidueSubset name="repack_only" selector="repack" >
      <RestrictToRepackingRLT/>
    </OperateOnResidueSubset>
  </TASKOPERATIONS>

  <MOVERS>
    <HBNetStapleInterface   name="hbnet"   monte_carlo="true"   scorefxn="commandline"
hb_threshold="-0.6"   min_networks_per_pose="1"   store_network_scores_in_pose="true"
minimize="false"                task_operations="extra_chi,incl_curr,fix,repack_only"
total_num_mc_runs="100000"/>
    <MultiplePoseMover name="limit_to_100" max_input_poses="100"/>
  </MOVERS>

  <PROTOCOLS>
    <Add mover="hbnet"/>
    <Add mover="limit_to_100"/>
  </PROTOCOLS>

</ROSETTASCRIPTS>
```

run.xml:

```
<ROSETTASCRIPTS>

  <RESIDUE_SELECTORS>
    <ResiduePDBInfoHasLabel name="interface" property="INTERFACE" />
    <ResiduePDBInfoHasLabel name="design"    property="DESIGN" />
    <ResiduePDBInfoHasLabel name="repack"    property="REPACK" />
    <ResiduePDBInfoHasLabel name="fixed"     property="FIXED" />
    <ResiduePDBInfoHasLabel name="hbnet"     property="HBNet" />

    <ResiduePropertySelector name="polar" properties="POLAR,CHARGED" logic="or_logic"
/>
    <And name="polar_at_designable_interface" selectors="polar,design"/>
  </RESIDUE_SELECTORS>

  <TASKOPERATIONS>
    <ExtraRotamersGeneric name="extra_chi" ex1="1" ex2="1" />
    <IncludeCurrent name="incl_curr" />
    <SetIGType name="linmem_ig" lin_mem_ig="true" />

    <OperateOnResidueSubset name="fix" selector="fixed" >
      <PreventRepackingRLT/>
    </OperateOnResidueSubset>

    <OperateOnResidueSubset name="fix_hbnet" selector="hbnet" >
      <PreventRepackingRLT/>
    </OperateOnResidueSubset>

    <OperateOnResidueSubset name="repack_only" selector="repack" >
      <RestrictToRepackingRLT/>
    </OperateOnResidueSubset>
  </TASKOPERATIONS>

  <FILTERS>
    <BuriedUnsatHbonds     name="buh_sc_heavy"     report_sc_heavy_atom_unsats="true"
cutoff="99999" residue_selector="interface" use_ddG_style="true"/>
    <BuriedUnsatHbonds     name="buh_bb_heavy"     report_bb_heavy_atom_unsats="true"
cutoff="99999" residue_selector="interface" use_ddG_style="true"/>
    <BuriedUnsatHbonds   name="buh_H"   report_nonheavy_unsats="true"   cutoff="99999"
residue_selector="interface" use_ddG_style="true"/>
  </FILTERS>

  <SIMPLE_METRICS>
    <SequenceMetric name="des_seq" residue_selector="design" />
    <SelectedResidueCountMetric                                        name="n_polar"
residue_selector="polar_at_designable_interface" />
    <SelectedResidueCountMetric name="n_designable" residue_selector="design" />
  </SIMPLE_METRICS>

  <MOVERS>
    <FastDesign    name="design"    relaxscript="%%script%%"    scorefxn="commandline"
task_operations="extra_chi,incl_curr,linmem_ig,fix,fix_hbnet,repack_only"/>
```

```
    <InterfaceAnalyzerMover name="IfaceAnalyzer" scorefxn="commandline" packstat="1"
interface_sc="1" pack_input="0" pack_separated="1" jump="1" tracer="false" />
    <FilterReportAsPoseExtraScoresMover     name="BUNS1"     report_as="BUNS_sc_heavy"
filter_name="buh_sc_heavy"/>
    <FilterReportAsPoseExtraScoresMover     name="BUNS2"     report_as="BUNS_bb_heavy"
filter_name="buh_bb_heavy"/>
    <FilterReportAsPoseExtraScoresMover        name="BUNS3"        report_as="BUNS_H"
filter_name="buh_H"/>

    <RunSimpleMetrics name="rsm1" metrics="des_seq" prefix="seq_" />
    <RunSimpleMetrics name="rsm2" metrics="n_polar" prefix="pol_" />
    <RunSimpleMetrics name="rsm3" metrics="n_designable" prefix="des_" />
  </MOVERS>

  <PROTOCOLS>
    <Add mover="design"/>

    <Add mover="IfaceAnalyzer"/>
    <Add mover="BUNS1"/>
    <Add mover="BUNS2"/>
    <Add mover="BUNS3"/>

    <Add mover="rsm1"/>
    <Add mover="rsm2"/>
    <Add mover="rsm3"/>
  </PROTOCOLS>

</ROSETTASCRIPTS>
```

REFERENCES

1. Leaver-Fay, A. *et al*. Rosetta3: An Object-Oriented Software Suite for the Simulation and Design of Macromolecules. *Methods Enzymol.* **487**, 545–574 (2011).

2. Stranges, P. B. & Kuhlman, B. A comparison of successful and failed protein interface designs highlights the challenges of designing buried hydrogen bonds. *Protein Sci.* **22**, 74–82 (2013).

3. Maguire, J. B., Boyken, S. E., Baker, D. & Kuhlman, B. Rapid Sampling of Hydrogen Bond Networks for Computational Protein Design. *J. Chem. Theory Comput.* (2018) doi:10.1021/acs.jctc.8b00033.

4. Tyka, M. D. *et al*. Alternate states of proteins revealed by detailed energy landscape mapping. *J Mol Biol* **405**, 607–618 (2011).

5. Fleishman, S. J. *et al*. RosettaScripts: A Scripting Language Interface to the Rosetta Macromolecular Modeling Suite. (2011) doi:10.1371/journal.pone.0020161.

6. Rohl, C. A., Strauss, C. E. M., Misura, K. M. S. & Baker, D. Protein Structure Prediction Using Rosetta. *Methods Enzymol.* **383**, 66–93 (2004).

7. Kuhlman, B. *et al*. Design of a Novel Globular Protein Fold with Atomic-Level Accuracy. *Science (80-.).* **302**, 1364–1368 (2003).

8. Leaver-Fay, A. *et al*. *Scientific benchmarks for guiding macromolecular energy function improvement. Methods in Enzymology* vol. 523 (Elsevier Inc., 2013).

9. O'Meara, M. J., Leaver-Fay, A. & Kuhlman, B. A Combined Covalent-Electrostatic Model of Hydrogen Bonding Improves Structure Prediction with Rosetta. *J Chem Theory Comput* **6**, 356–372 (2015).

10. Alford, R. F. *et al*. The Rosetta All-Atom Energy Function for Macromolecular Modeling and Design. *J. Chem. Theory Comput.* **13**, 3031–3048 (2017).

11. InterGroupInterfaceByVector Documentation. https://www.rosettacommons.org/docs/latest/scripting_documentation/RosettaScripts/ResidueSelectors/ResidueSelectors#residueselectors_conformation-dependent-residue-selectors_intergroupinterfacebyvector.

12. Lawrence, M. C. & Colman, P. M. Shape complementarity at protein/protein interfaces. *Journal of Molecular Biology* vol. 234 946–950 (1993).

13. Kuroda, D. & Gray, J. J. Shape complementarity and hydrogen bond preferences in protein-protein interfaces: implications for antibody modeling and protein-protein docking. *Bioinformatics* **32**, 2451–6 (2016).

14. BuriedUnsatHbonds Documentation. https://www.rosettacommons.org/docs/latest/scripting_documentation/RosettaScripts/Filters/filter_pages/BuriedUnsatHbondsFilter.

15. Guffy, S. L., Teets, F. D., Langlois, M. I. & Kuhlman, B. Protocols for Requirement-Driven Protein Design in the Rosetta Modeling Program. *J. Chem. Inf. Model.* **58**, 895–901 (2018).

CHAPTER 5: Using a Deep Neural Network to Improve Low-Resolution Modeling of Protein-Protein Interactions Prior to Design

5.1 Introduction

5.1.1 Background

Artificial neural networks have stormed the field of protein modeling in recent years, perhaps most notably with AlphaFold's success in CASP13.[1] While many of these developments are in the field of protein structure prediction,[1–11] de novo protein design is seeing great advances as well.[12–15] Scientists use machine learning to study underlying protein backbone patterns in protein crystal structures, then generate novel native-like backbones that can be stabilized through sequence design programs like Rosetta.[16,17] This backbone-centric modeling is very powerful in terms of sampling large areas of *de novo* design space in a short amount of time. However, there are some design cases that still have room for improvement.

To date, all neural networks used for protein modeling either require each residue position to declare an amino acid identity or require each residue position to completely abandon its identity. An example of the former is AlphaFold, which assigns each residue an amino acid identity and then predicts how that protein folds based on its amino acids.[1] An example of the latter is SCUBA, which only considers the quality of a protein conformation by its backbone.[18] There is no explicit tool for hybrid cases such as docking prior to one-sided interface design, in which one protein's residues are strictly fixed-sequence and the other protein is allowed to make mutations to improve binding strength. The best a user can do is use one of the two types of existing methods and hope that their docking tool works despite operating with incorrect or incomplete information.

This is an imperfect solution to a big problem, namely protein interface design. In this paper we present a deep neural network that utilizes the complete information about each residue's downstream amino acid identity options.

5.1.2 Project Description

To accomplish this, we chose to build a system in the context of the Rosetta protein modeling software suite.[16] Rosetta has recently been used in successful one-sided interface design projects,[19,20] so we expected it to be a good medium for us to accomplish this task. Traditionally, users employ score3,[21] Rosetta's low-resolution energy function, to model their proteins while they are in a low-resolution representation (meaning not every atom in the sidechain is modeled) in backbone-conformation-sampling cases like docking and loop modeling. score3 requires each residue to have an amino acid identity, so designable residues are generally represented as a mid-sized, nonpolar amino acid like valine. This is a drawback because it deprives Rosetta of important information. Rosetta expects each residue to end up having a mid-sized, nonpolar sidechain and makes structural decisions around that assumption. We plan to replace score3 with a system that tells Rosetta which residues are designable and, for the designable residues, which amino acids are available at each position.

Unlike score3, our new tool MOUSE (Model Of Ultimate Surface Energy) will not be pairwise-decomposable;[21] it will assign a score to each surface residue by collectively considering every other residue in the immediate environment. We chose to implement this by performing ray casting. Each residue is represented by a single sphere and casts rays in all directions from the center of its sphere (roughly the center of mass for that residue). Each ray travels in space until it hits a neighboring residue's sphere, at which point it returns information regarding the residue it hit. This ray information is combined with information about the "source" residue and together they are fed into a neural network which predicts the final energy of the source residue after design (in Rosetta Energy Units, REU).

Our expectation is that this additional information about downstream amino acid identities will allow Rosetta to better establish favorable conformations in early pre-design stages such as docking and backbone sampling.

5.2 Methods

5.2.1 Ray Casting

A coordinate frame is defined for each residue such that the residue has a well-defined latitude and longitude framework. A ray is cast every 10 degrees longitude for all 360 degrees (resulting in 36 longitudinal values) and every 10 degrees latitude for 180 degrees (resulting in 19 latitudinal values from -90 to +90 degrees). To prevent redundancy and increase resolution at the poles, no rays are cast at directly +90 or -90 degrees latitude, the north and south poles respectively. The latitudinal values of +90 and -90 degrees are replaced with +85 and -85 degrees (technically only spanning 170 degrees instead of 180).

During ray casting, each residue is represented by a sphere 3 Å in radius centered at the residue's CB atom (all residues are temporarily converted to valine to ensure uniform CB atom placement). Each ray travels from the center of the residue being scored until it hits another residue or travels the maximum distance of 10 Å. Each ray returns 27 values when it hits another residue, some of which are illustrated in Figure 5.1.

Values 1-20 are each mapped to an amino acid identity (1:ALA, 2:CYS, …, 20:TYR) and either adopt the value of 1 or 0. A value of 1 means that the residue position hit by the ray is allowed to adopt the amino acid being represented, either by mutating to that amino acid or being that amino acid natively. In this system, residues that are fully designable will return twenty 1's when hit. Conversely, residues that are fixed sequence will return one 1 (at the position that represents the native amino acid for that residue) and nineteen 0's.

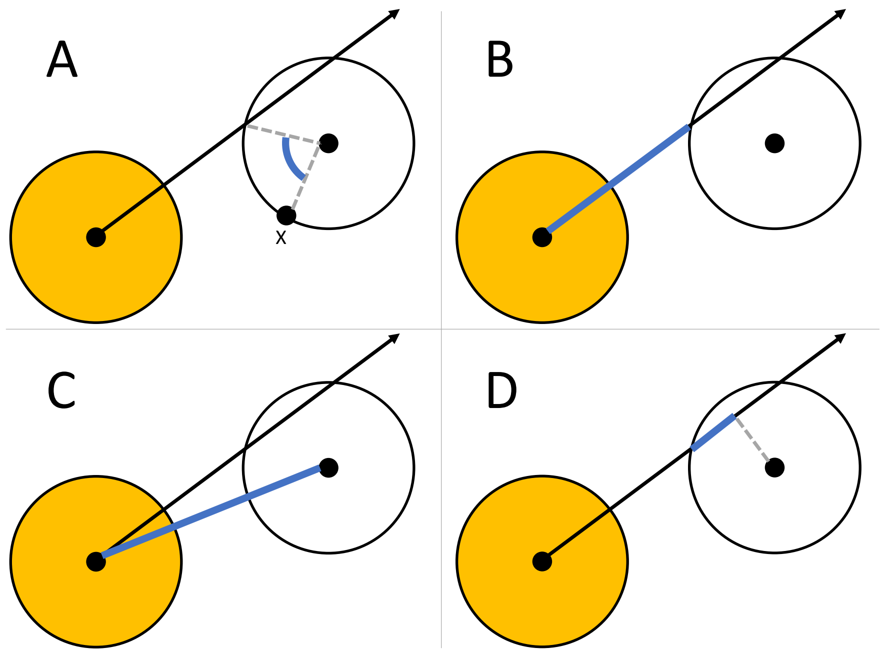


Figure 5.1 *Visualization of ray casting geometries that are fed into the neural network. In this case, the residue being evaluated by the neural network is shown in yellow, a neighboring residue that is hit by the ray is shown in white, the ray being cast is shown in black, and the geometric feature being illustrated is shown in blue. The black circles are centers of the spheres. (A) angle between ray intersection point of the sphere and some arbitrary atom X. (B) Distance that the ray travels. (C) Distance between the centers of the two spheres. (D) Distance between the ray intersection point and the closest point on the ray to the neighboring sphere's center.*

Values 21, 22, and 23 represent different variations of the angle shown in Figure 5.1.A. The angle is measured where the atom labeled X is the CA atom, C atom, and N atom. The other two vertices of the angle are the center of the sphere and the intersection point of the ray with the sphere's surface.

Value 24 is simply the distance traveled between the center of the sphere for the residue begin evaluated and the intersection point of the ray with the sphere that is hit (Figure 5.1.B).

Similarly, Value 25 is the distance between the center of the sphere for the residue begin evaluated and the center of the sphere that is hit by the ray (Figure 5.1.C). Value 26 is the distance between the ray's intersection point and the point on the ray that is closest to the hit sphere's center (Figure 5.1.D). The 27th and final value is a 1 if the residue being hit is an immediate sequence neighbor to the residue being evaluated (residue that is casting the ray) and a 0 otherwise.

5.2.1.1 Additional Data Collected

MOUSE also considered 26 values regarding the residue being scored. Since these values are constant for each ray, they are not included in the ray information and are inserted into the neural network separately. Values 1-20 are identical to the first 20 values that each ray returns, except the logic is applied to the residue casting the rays instead of the residue being hit. Value 21 is 1 if the residue is a C-terminus and 0 otherwise. Likewise, value 22 is 1 if the residue is a N-terminus and 0 otherwise. Values 23-26 are the sine and cosine of the residue's phi and psi angles.

5.2.2 MOUSE Implementation

We implemented two MOUSE neural networks: generation 1 and generation 2. The first generation is roughly 5 times slower than the second generation due to its neural network being several times larger. The specific architectures of each network are shown in sections 5.5.1 and 5.5.2. Generation 1 had 741,046 trainable parameters and generation 2 had 66,202 trainable parameters.

The networks each have two input tensors and one output tensor. One input tensor holds the result of the ray casting calculations and has a dimension 36 x 19 x 27 (the data is provided as 18468 x 1 and reshaped immediately inside the network). The other input tensor holds various metrics about the residue being scored (as described in section 5.2.1.1) and has a dimension of 26 x 1. The output tensor holds just a single element and that lone value represents the normalized predicted score for the residue being evaluated. Generation 1 uses Equation 5.1 to normalize its Rosetta energy prediction (marked as x) and Generation 2 uses Equation 5.2. The goal of these nonlinear normalizations methods is to make mispredictions in the -10 to -3 REU range more sensitive than mispredictions in the 100 to 500 REU range because the former range is where the majority of residues fall after FastDesign. The normalization method was changed after Generation 1 in an effort to be more elegant.

$$f_{norm}(x) = \max(\ \ln(\ \ln(\ \max(10 + x, 1.00001)\ )\ ) - 1, -1.4\ ) \qquad \text{Equation 5.1}$$

$$f_{norm}(x) = e^{x/-15.0} - 1 \qquad \text{Equation 5.2}$$

5.2.3 Neural Network Training

Both generations were trained using the Adam[22] optimizer with a mean squared error loss function and a learning rate of 0.001. Various learning rate schedules were tested but none resulted in improvement. Our batch sizes alternated between 32 and 64 depending on GPU memory restrictions. Network creation and training were executed using Keras[23] with a TensorFlow[24] backend. The test set loss metric was measured by Keras during training. Training was manually terminated when we empirically determined the testing loss had either plateaued or started regressing. Due to our abundance of training data, both generation 1 and generation 2 plateaued before the end of the first epoch.

5.2.3.1 Training Data

We divided the top8000 protein structure library[25] into two segments of size 5000 (training set) and 3000 (testing set). To generate data for either set, we randomly drew two single chains from the set and randomly docked them together. We then assigned each residue position on the surface of both proteins a random set of amino acids it could mutate to by drawing 20 random Booleans for each position, each Boolean determining if its respective amino acid was allowed at that position. To generate the expected output for the neural network, the protein underwent Rosetta's fixed-backbone rotamer substitution protocol[21] using a modified score function with a decreased repulsive weight and modified reference energies. This score function modification was intended to mimic the conditions of the first round of the PolarDesign2019 variant of FastDesign (see Chapter 3). The fixed-backbone rotamer substitution protocol was performed three times and the average energy for each residue over the three runs was taken. This entire process was repeated roughly 1,000,000 times in an effort to prevent the lack of training data from limiting our success.

5.2.4 One-Sided Interface Design Benchmark

For this test, we performed 5 production runs of SEWING[26], each run giving us between 100 and 2500 structures. The structures are composed of one *de novo* SEWING chain bound to a native G-alpha(q) binder as described in section 3.6.1. For the purposes of this test, the SEWING chain was allowed to make unlimited mutations and the native binder was restricted to its native sequence.

We evaluated each structure with several low-resolution metrics: (1) score3,[21] (2) a geometric hash-based term called "motif score"[27] that is traditionally used by SEWING and other backbone-generating protocols,[26] and (3) several variants of MOUSE as described below. For score3, residue positions that could be designed (in which mutations are legal) were represented as valine, as is commonly done by Rosetta users. We then fed these structures into FastDesign using an identical technique as is described in Chapter 4 (using ref2015[28] with PolarDesign2019, no MC HBNet[29]) to generate the high-resolution scores for each structure. Note that the MOUSE scores, the score3 scores, and the high-resolution scores were normalized by dividing by the number of residues in the entire protein complex. Motif scores are inherently normalized so they were left untouched. For each of the 5 production runs, we measured the Pearson correlation between the high-resolution score after design and each of the low-resolution metrics.

We performed this benchmark with four variants of MOUSE. Generation 1 and Generation 2 were both run twice. The first run was performed normally, recognizing that the SEWING chain was allowed to make mutations and the binder had a fixed sequence. The second run (labeled with an asterisk in Figure 5.2) had incomplete information; MOUSE was incorrectly told that both chains could make unlimited mutations. The difference in performance between the two runs for a given generation will represent the amount of performance gained by using the sequence information of the native binder; information that all other existing protein-design machine-learning models ignore.

For each of the five production runs of SEWING, we measured the Pearson correlations between the various low-resolution terms (score3, motif score, and various MOUSE terms) and the high-resolution post-design energy determined by FastDesign. These correlations are represented as box-and-whisker plots in Figure 5.2.

5.3 Results and Discussion

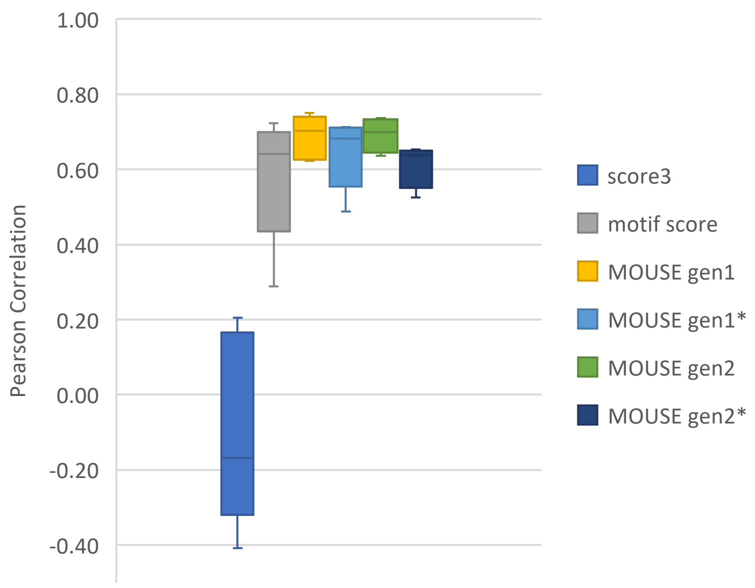


Figure 5.2 *Correlation distributions between several low-resolution metrics and the final high-resolution score after FastDesign. MOUSE runs with incomplete amino acid information are denoted with asterisks (*).*

We accumulated the results of 5 productions runs of SEWING, each with between 100 and 2500 generated decoys. We evaluated each set with a handful of low-resolution metrics, then performed high-resolution design on them. The correlation between the low-resolution metrics and the final high-resolution score after design was measure for each production run, and the correlations across the 5 productions runs are represented in Figure 5.2.

Rosetta's default low-resolution scoring function, score3, has a poor correlation. To counter this, SEWING developers started using a geometric hash-based Motif Score, which we see greatly outperforms score3. Encouragingly, we see that both generations of MOUSE outperform SEWING's own Motif Score for ranking SEWING designs.

MOUSE's advantage may be in part due to the Motif Score treating all residues as designable, thus being unable to use all of the information that MOUSE has available to it. We ran gen1* and gen2* of MOUSE to test this theory, both which operated under the false assumption that all residues in the protein were able to mutate without restriction. This is the same assumption that the motif score operates under. Figure 5.2 shows that gen1* and gen2* dip in quality compared to gen1 and gen2 and are instead more comparable to the quality of the motif score. This suggests that the information about the fixed-sequence residues' amino acid identities can measurably improve the accuracy of low-resolution score functions. It also suggests that this information is perhaps the only property of MOUSE that makes it outperform the motif score. gen1* and, to a lesser extent, gen2* are still slight improvements over the motif score but not by much.

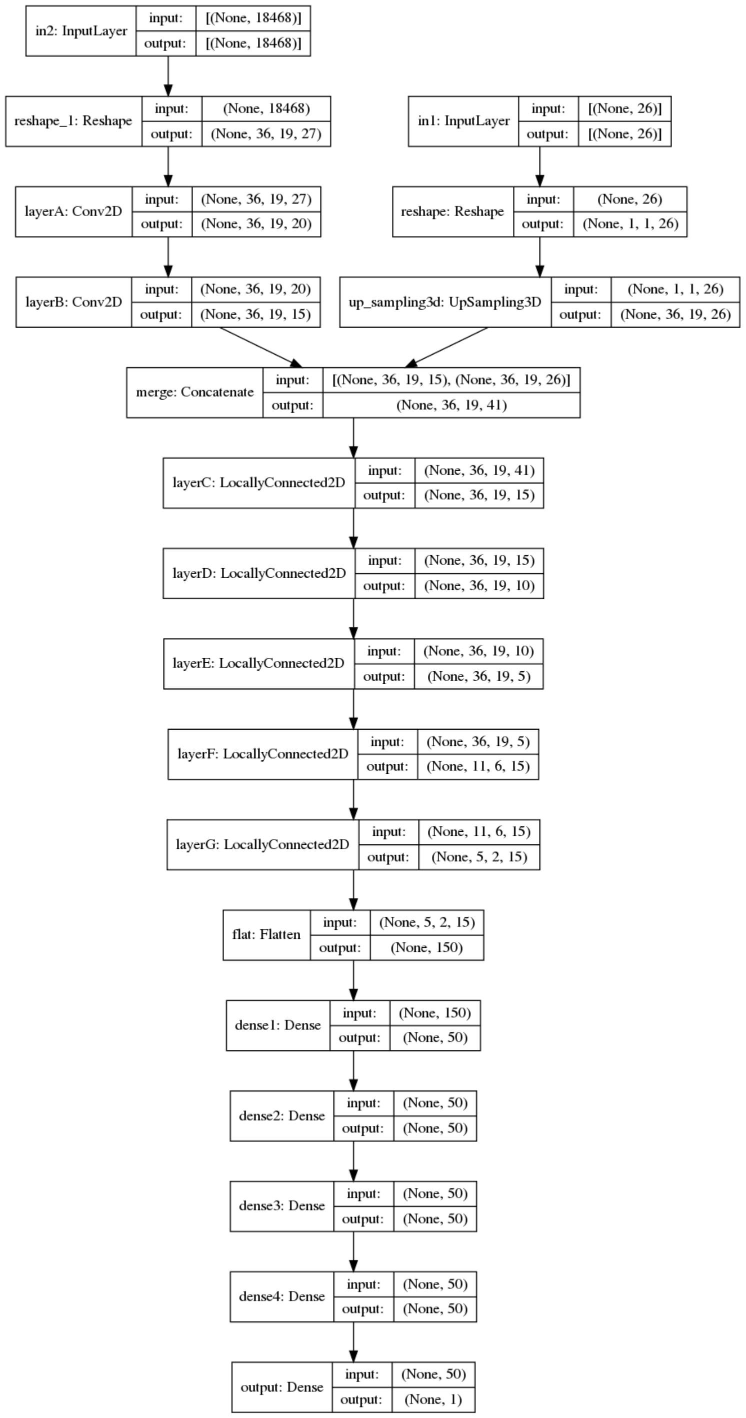
5.4 Conclusion and Future Work

In this chapter we showed that MOUSE is better at discriminating low-resolution interface decoys than the current gold standard of the motif score. This can be useful for saving significant amounts of computer time by eliminating candidate docking conformations without running expensive high-resolution packing protocols on them.

The benchmark shown in this chapter is a promising preliminary result, but we have plans to test MOUSE further. These plans include using MOUSE as a filter between docking and design phases of interface design for a project that will test these interfaces experimentally. We also plan to test MOUSE's ability to guide docking trajectories by providing a more accurate energy landscape. Rosetta's docking protocols all still use score3 as their default score function and we saw in Figure 5.2 just how poor score3 correlates with final design quality, so we will see if MOUSE can play a role there.

5.5 Supplemental Information

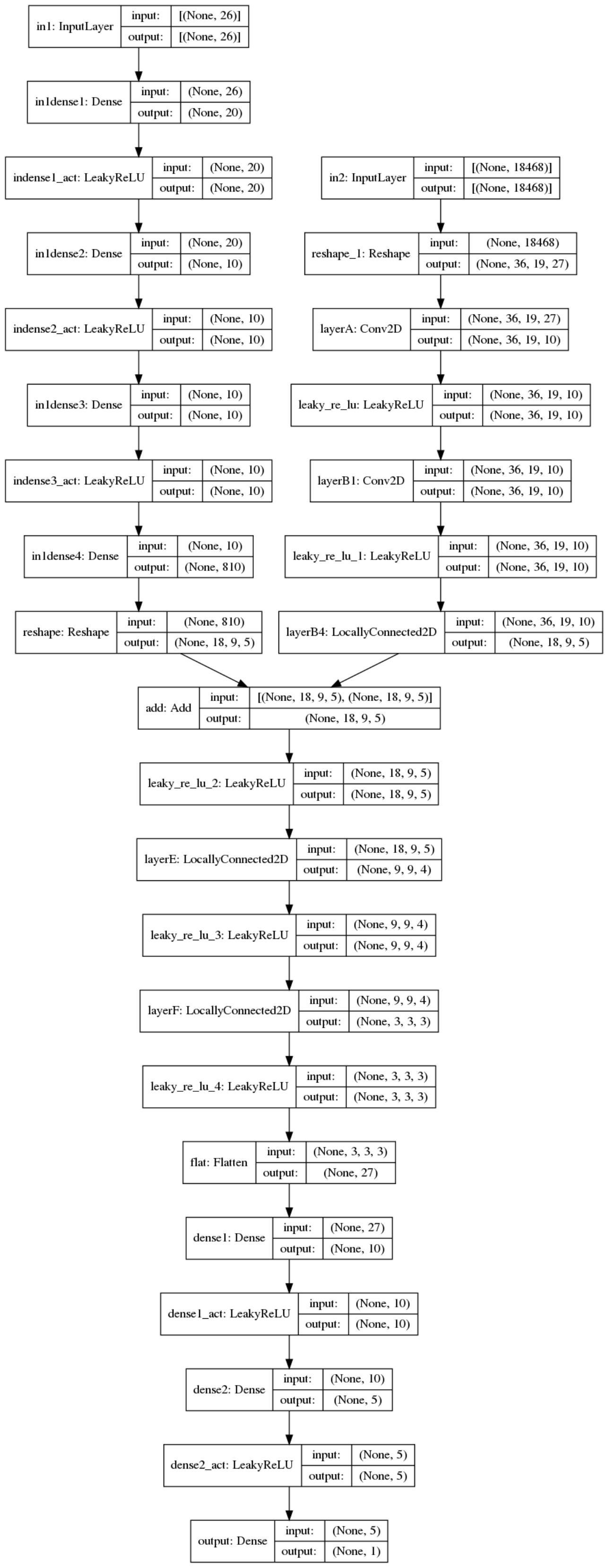
5.5.1 MOUSE Generation 1 Network Architecture



in2: InputLayer | input: [(None, 18468)] | output: [(None, 18468)]
reshape_1: Reshape | input: (None, 18468) | output: (None, 36, 19, 27)
in1: InputLayer | input: [(None, 26)] | output: [(None, 26)]
layerA: Conv2D | input: (None, 36, 19, 27) | output: (None, 36, 19, 20)
reshape: Reshape | input: (None, 26) | output: (None, 1, 1, 26)
layerB: Conv2D | input: (None, 36, 19, 20) | output: (None, 36, 19, 15)
up_sampling3d: UpSampling3D | input: (None, 1, 1, 26) | output: (None, 36, 19, 26)
merge: Concatenate | input: [(None, 36, 19, 15), (None, 36, 19, 26)] | output: (None, 36, 19, 41)
layerC: LocallyConnected2D | input: (None, 36, 19, 41) | output: (None, 36, 19, 15)
layerD: LocallyConnected2D | input: (None, 36, 19, 15) | output: (None, 36, 19, 10)
layerE: LocallyConnected2D | input: (None, 36, 19, 10) | output: (None, 36, 19, 5)
layerF: LocallyConnected2D | input: (None, 36, 19, 5) | output: (None, 11, 6, 15)
layerG: LocallyConnected2D | input: (None, 11, 6, 15) | output: (None, 5, 2, 15)
flat: Flatten | input: (None, 5, 2, 15) | output: (None, 150)
dense1: Dense | input: (None, 150) | output: (None, 50)
dense2: Dense | input: (None, 50) | output: (None, 50)
dense3: Dense | input: (None, 50) | output: (None, 50)
dense4: Dense | input: (None, 50) | output: (None, 50)
output: Dense | input: (None, 50) | output: (None, 1)

Note: activation layers are not shown, but ReLU is present after every layer except “in1”, “in2”, “up_sampling3d”, “merge”, “flat”, and “output”.

5.5.2 MOUSE Generation 2 Network Architecture



in1: InputLayer input: [(None, 26)] output: [(None, 26)]
in1dense1: Dense input: (None, 26) output: (None, 20)
indense1_act: LeakyReLU input: (None, 20) output: (None, 20)
in1dense2: Dense input: (None, 20) output: (None, 10)
indense2_act: LeakyReLU input: (None, 10) output: (None, 10)
in1dense3: Dense input: (None, 10) output: (None, 10)
indense3_act: LeakyReLU input: (None, 10) output: (None, 10)
in1dense4: Dense input: (None, 10) output: (None, 810)
reshape: Reshape input: (None, 810) output: (None, 18, 9, 5)
in2: InputLayer input: [(None, 18468)] output: [(None, 18468)]
reshape_1: Reshape input: (None, 18468) output: (None, 36, 19, 27)
layerA: Conv2D input: (None, 36, 19, 27) output: (None, 36, 19, 10)
leaky_re_lu: LeakyReLU input: (None, 36, 19, 10) output: (None, 36, 19, 10)
layerB1: Conv2D input: (None, 36, 19, 10) output: (None, 36, 19, 10)
leaky_re_lu_1: LeakyReLU input: (None, 36, 19, 10) output: (None, 36, 19, 10)
layerB4: LocallyConnected2D input: (None, 36, 19, 10) output: (None, 18, 9, 5)
add: Add input: [(None, 18, 9, 5), (None, 18, 9, 5)] output: (None, 18, 9, 5)
leaky_re_lu_2: LeakyReLU input: (None, 18, 9, 5) output: (None, 18, 9, 5)
layerE: LocallyConnected2D input: (None, 18, 9, 5) output: (None, 9, 9, 4)
leaky_re_lu_3: LeakyReLU input: (None, 9, 9, 4) output: (None, 9, 9, 4)
layerF: LocallyConnected2D input: (None, 9, 9, 4) output: (None, 3, 3, 3)
leaky_re_lu_4: LeakyReLU input: (None, 3, 3, 3) output: (None, 3, 3, 3)
flat: Flatten input: (None, 3, 3, 3) output: (None, 27)
dense1: Dense input: (None, 27) output: (None, 10)
dense1_act: LeakyReLU input: (None, 10) output: (None, 10)
dense2: Dense input: (None, 10) output: (None, 5)
dense2_act: LeakyReLU input: (None, 5) output: (None, 5)
output: Dense input: (None, 5) output: (None, 1)

5.5.3 Visualization of What MOUSE "Sees"

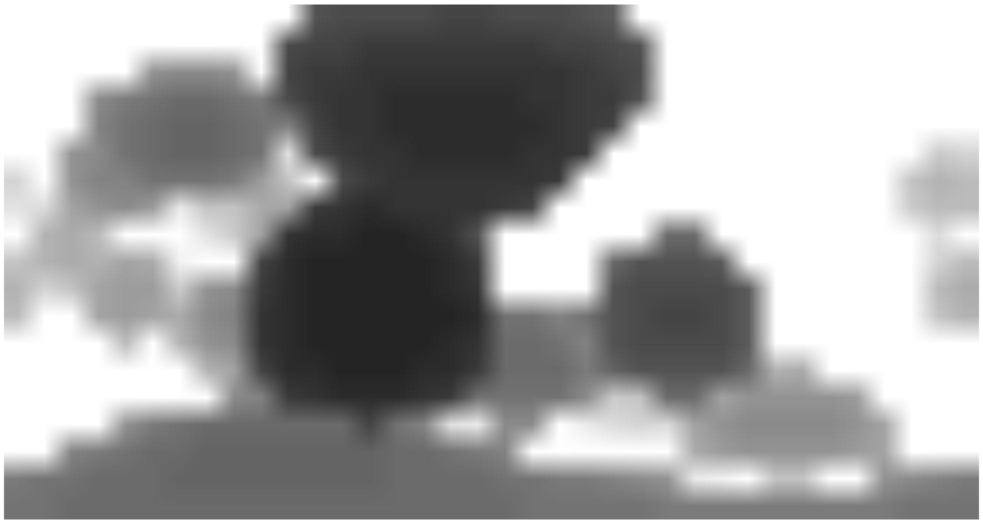


Figure 5.S1 Realistic visualization of how MOUSE sees distance

Shown above is a partial representation of the ray-tracing data fed into MOUSE. This is a Mercator projection, with the x-axis corresponding to longitude and y-axis with latitude. Only ray-travel distance is illustrated, with short distances darker than longer distances. Despite the distortion at the top and bottom caused by the Mercator projection, you can make out several circles. Each circle is a sphere that represents a neighboring residue. The residue being evaluated is at the source point for the ray tracing so it is not seen (in other words, it is the camera). The zig-zag patterns around the edges of the circles (partially blurred by the compression of this image) is simply due to the low resolution of the data collection; casting rays more densely would produce rounder circles.

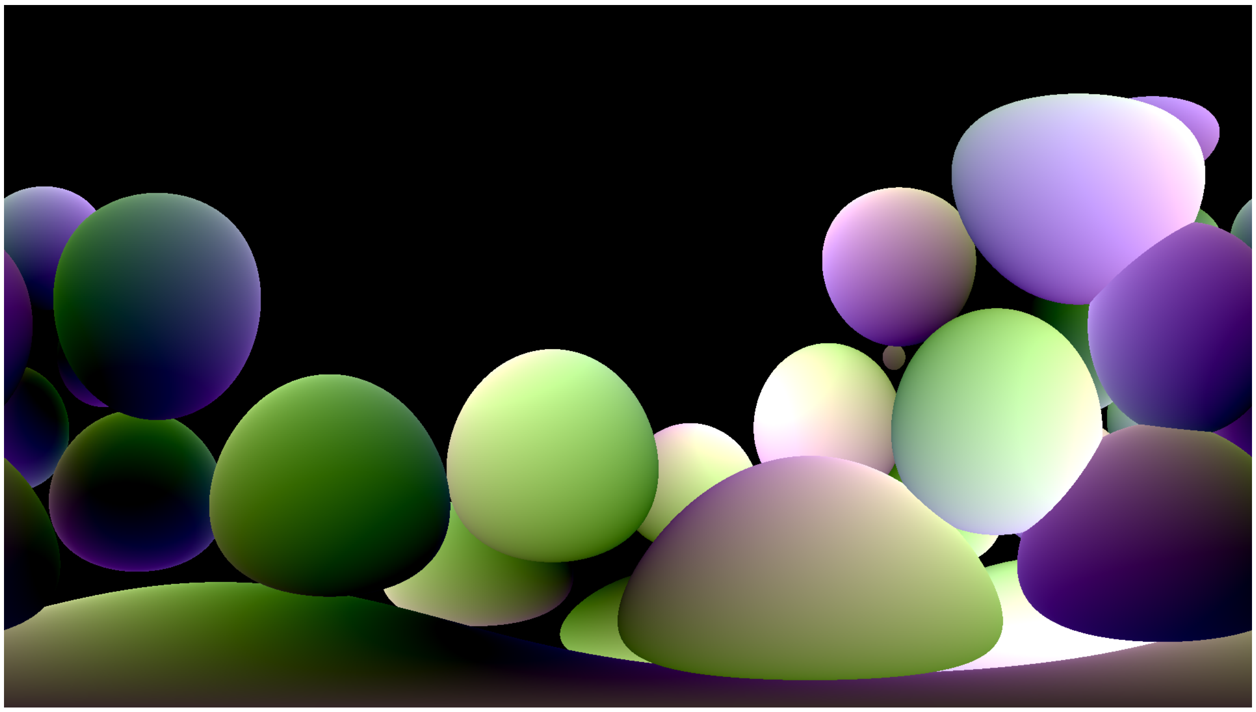


Figure 5.S2 HD visualization of how MOUSE sees relative orientation

Shown here is a higher-resolution set of ray-tracing results with a similar camera format to Figure 5.S1. Instead of showing distance, this image shows the relative orientations of neighboring residues. Values 21, 22, and 23 (as described in section 5.2.1) are represented by the red, green, and blue channels of this image respectively. As a result, the black poles of each sphere illustrated where the backbone is centered and the white poles show where the sidechains are.

REFERENCES

1. Senior, A. W. *et al*. Protein structure prediction using multiple deep neural networks in the 13th Critical Assessment of Protein Structure Prediction (CASP13). *Proteins Struct. Funct. Bioinforma*. **87**, 1141–1148 (2019).

2. Zhou, J. & Troyanskaya, O. G. Deep Supervised and Convolutional Generative Stochastic Network for Protein Secondary Structure Prediction. (2014).

3. Yang, J. *et al*. Improved protein structure prediction using predicted inter-residue orientations. *bioRxiv* 846279 (2019) doi:10.1101/846279.

4. Sønderby, S. K. & Winther, O. Protein Secondary Structure Prediction with Long Short Term Memory Networks. (2014).

5. Ma, Y., Liu, Y. & Cheng, J. Protein secondary structure prediction based on data partition and semi-random subspace method. *Sci. Rep*. **8**, (2018).

6. Torrisi, M., Kaleel, M. & Pollastri, G. Porter 5: state-of-the-art ab initio prediction of protein secondary structure in 3 and 8 classes. *bioRxiv* 289033 (2018) doi:10.1101/289033.

7. Xu, J. Distance-based Protein Folding Powered by Deep Learning. (2018).

8. Gao, M., Zhou, H. & Skolnick, J. DESTINI: A deep-learning approach to contact-driven protein structure prediction. *Sci. Rep*. **9**, (2019).

9. AlQuraishi, M. End-to-End Differentiable Learning of Protein Structure. *Cell Syst*. **8**, 292-301.e3 (2019).

10. Asgari, E., Poerner, N., McHardy, A. C. & Mofrad, M. R. K. DeepPrime2Sec: Deep Learning for Protein Secondary Structure Prediction from the Primary Sequences. *bioRxiv* 705426 (2019) doi:10.1101/705426.

11. Greener, J. G., Kandathil, S. M. & Jones, D. T. Deep learning extends de novo protein modelling coverage of genomes using iteratively predicted structural constraints. *Nat. Commun*. **10**, (2019).

12. Yang, K. K., Wu, Z. & Arnold, F. H. Machine-learning-guided directed evolution for protein engineering. *Nature Methods* vol. 16 687–694 (2019).

13. Sabban, S. & Markovsky, M. RamaNet: Computational De Novo Protein Design using a Long Short-Term Memory Generative Adversarial Neural Network. *bioRxiv* 671552 (2019) doi:10.1101/671552.

14. Ingraham, J., Garg, V. K., Barzilay, R. & Csail, T. J. *GENERATIVE MODELS FOR GRAPH-BASED PROTEIN DESIGN*.

15. O'Connell, J. *et al*. SPIN2: Predicting sequence profiles from protein structures using deep neural networks. *Proteins Struct. Funct. Bioinforma.* **86**, 629–633 (2018).

16. Leaver-Fay, A. *et al*. Rosetta3: An Object-Oriented Software Suite for the Simulation and Design of Macromolecules. *Methods Enzymol.* **487**, 545–574 (2011).

17. Kaufmann, K. W., Lemmon, G. H., Deluca, S. L., Sheehan, J. H. & Meiler, J. Practically Useful: What the ROSETTA Protein Modeling Suite Can Do for You. doi:10.1021/bi902153g.

18. Huang, B., Xu, Y. & Liu, H. Combining statistical and neural network approaches to derive energy functions for completely flexible protein backbone design. *bioRxiv* 673897 (2019) doi:10.1101/673897.

19. Tinberg, C. E. & Khare, S. D. Computational Design of Ligand Binding Proteins. in *Methods in molecular biology (Clifton, N.J.)* vol. 1529 363–373 (2017).

20. Leaver-Fay, A. *et al*. Computationally Designed Bispecific Antibodies using Negative State Repertoires. *Structure* **24**, 641–651 (2016).

21. Kuhlman, B. *et al*. Design of a Novel Globular Protein Fold with Atomic-Level Accuracy. *Science (80-.).* **302**, 1364–1368 (2003).

22. Kingma, D. P. & Ba, J. L. Adam: A method for stochastic optimization. in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (International Conference on Learning Representations, ICLR, 2015).

23. Keras. keras.io.

24. Tensorflow. https://www.tensorflow.org.

25. Keedy, D. A. *et al*. 8000 Filtered Structures. http://kinemage.biochem.duke.edu/databases/top8000.php (2012).

26. Guffy, S. L., Teets, F. D., Langlois, M. I. & Kuhlman, B. Protocols for Requirement-Driven Protein Design in the Rosetta Modeling Program. *J. Chem. Inf. Model.* **58**, 895–901 (2018).

27. Marze, N. A., Roy Burman, S. S., Sheffler, W. & Gray, J. J. Efficient flexible backbone protein–protein docking for challenging targets. *Bioinformatics* **34**, 3461–3469 (2018).

28. Alford, R. F. *et al*. The Rosetta All-Atom Energy Function for Macromolecular Modeling and Design. *J. Chem. Theory Comput.* **13**, 3031–3048 (2017).

29. Maguire, J. B., Boyken, S. E., Baker, D. & Kuhlman, B. Rapid Sampling of Hydrogen Bond Networks for Computational Protein Design. *J. Chem. Theory Comput.* (2018) doi:10.1021/acs.jctc.8b00033.