Enhanced Algorithms for Analysis and Design of Nucleic Acid Reaction Pathways

Thesis by
Nicholas James Porubsky

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2020
Defended September 16, 2019

Acknowledgements

First, I would like to thank my advisor, Niles Pierce. Time and again he has helped me realize that research is failure punctuated by success and therefore persistence and pivoting to alternatives pay off. I would like to thank my committee chair, Erik Winfree, for his expert knowledge in algorithms adjacent to those of our lab as well as his role in starting and sending off the Molecular Programming Project, of which I've been proud to be a small part. I would like to thank my remaining committee members, Rustem Ismagilov and Zhen-Gang Wang, for pushing me to take responsbility for seeking out the prerequisite knowledge necessary to work on the algorithms in this thesis and to develop a physical intuition for the systems I work on.

I thank the members of the Pierce lab who I've worked with over the years. First and foremost, I thank Mark Fornace, for being my longest collaborator and co-author on our algorithmic and model improvements paper. He is both an excellent scientist as well as an excellent software engineer, a rare breed to be sure. Without his initiative in beginning our rewrite of the NUPACK thermodynamics code, crafting the algorithm in the last chapter of this thesis would not have been possible. Secondly, I'd like to thank Brian Wolfe, my predecessor in the lineage of nucleic acid design algorithm developers. His work with Niles enabled design of test tubes of nucleic acid systems (a game-changing improvement to sequence design) as well as much of the algorithmic and software work necessary for the multitube design project which we published together. I would also like to thank the members of the lab who used the design software during development: Maayan, Lisa, Mikhail, Jining, and Zhewei. You all helped me find many bugs (fixed now!) and improve the documentation to the benefit of researchers around the world.

I would like to thank some of my classmates who became close friends: Ahmad, Mikey, and Jordan. Ahmad's ability to take the long view and laugh at the ridiculousness of it all have given me better perspective on the transient stresses in life. Mikey is quite possibly the most buoyant person I've met and his relentless enthusiasm for life has and continues to lift me up. Jordan can reliably sense stagnation and suggest a web of movies and video games to take your mind off things.

I would also like to thank two of my friends from back in Wisconsin, Becki and Lee, for their remote support through online gaming sessions and visits during trips back home. They helped me repeatedly wake up to the fact that despite the PhD process feeling all-encompassing at times (or even most of the time), there is indeed an outside world that complements the experiences of research.

I would like to thank my family. My parents, James and Andrea, stressed throughout my life the importance of striving for excellence in academic pursuits, and the mindset that engendered within me has led me to and through my PhD experience. Without direct experience, they intuited the need for higher education and, through the examples of their life, succeeded in instilling a strong work ethic in their three children and saw them successfully graduate from college. I also thank my siblings, Veronica and Dominic, for their patient ears and empathetic words while I've been pursuing my PhD. Having siblings who shared formative early life experiences to reminisce about has helped in dispelling solipsistic tendencies that emerged during the more stressful episodes of grad school.

Finally, I would like to thank my girlfriend, Rachel, who has been both consistently encouraging and understanding during the trials and tribulations of grad school (especially the last few months). Now that we're both moving on to new and exciting things, I look forward to having a partner to explore new horizons with and reminisce about the many good times we shared while at Caltech.

Abstract

Nucleic acids provide a powerful platform for programming at the molecular level. This is possible because the free energy of nucleic acid structures is dominated by the local interactions of base pairing and base pair stacking. The nearest neighbor secondary structure model implied by these energetics has enabled development of a set of algorithms for calculating thermodynamic quantities of nucleic acid sequences. Molecular programmers and synthetic biologists continue to extend their reach to larger, more complicated nucleic acid complexes, reaction pathways, and systems. This necessitates a focus on new algorithm development and efficient implementations to enable analysis and design of such systems.

Concerning analysis of nucleic acids, we collect seemingly diverse algorithms under a unified three-component dynamic programming framework consisting of: 1) recursions that specify the dependencies between subproblems and incorporate the details of the structural ensemble and the free energy model, 2) evaluation algebras that define the mathematical form of each subproblem, 3) operation orders that specify the computational trajectory through the dependency graph of subproblems. Changes to the set of recursions allows operation over the complex ensemble including coaxial and dangle stacking states, affecting all thermodynamic quantities. An updated operation order for structure sampling allows simultaneous generation of a set of structures sampled from the Boltzmann distribution in time that scales empirically sublinearly in the number of samples and leads to an order of magnitude or more speedup over repeated single-structure sampling.

For the problem of sequence design for reaction pathway engineering, we introduce an optimization algorithm to minimize the multitstate test tube ensemble defect, which simultaneously designs for reactant, intermediate, and product states along the reaction pathway (positive design) and against crosstalk interactions (negative design). Each of these on-pathway or crosstalk states is represented as a target test tube ensemble containing arbitrary numbers of on-target complexes, each with a target secondary structure and target concentration, and arbitrary numbers of off-target complexes, each with vanishing target concentration. Our test tube specification formalism enables conversion of a reaction pathway specification into a set of target test tubes. Sequences are designed subject to a set of hard constraints allowing specification of properties such as sequence composition, sequence complementarity, prevention of unwanted sequence patterns, and inclusion of biological sequences. We then extend this algorithm with soft constraints, enhancing flexibility through new constraint types and reducing design cost by up to two orders of magnitude in the most highly constrained cases. These soft constraints enable multiobjective design of the multitstate test tube ensemble defect simultaneously with heuristics for avoiding kinetic traps and equalizing reaction rates to further aid reaction pathway engineering.

Published Content and Contributions

[1] B. R. Wolfe*, N. J. Porubsky*, J. N. Zadeh, R. M. Dirks, and N. A. Pierce. “Constrained Multistate Sequence Design for Nucleic Acid Reaction Pathway Engineering”. In: *Journal of the American Chemical Society* 139.8 (2017), pp. 3134–3144. ISSN: 15205126. DOI: 10.1021/jacs.6b12693.
*These authors contributed equally. Algorithm development was begun by BRW and NAP following earlier work started by RMD and JNZ. BRW implemented the multitube framework and constraint satisfaction algorithm. The algorithm was further developed by BRW, NAP, and NJP. The general target test tube specification was developed by NAP and NJP. NJP implemented functionality to specify designs with target tube ensembles consisting of arbitrary sets of on-targets and off-targets to enable use of the target test tube specification. NJP formulated the target test tubes of the case study reaction pathways. Computational results were obtained by NJP. BRW, NJP, and NAP wrote the manuscript and supplementary information.

[2] M. E. Fornace*, N. J. Porubsky* and N. A. Pierce. “A Unified Dynamic Programming Framework for the Analysis of Interacting Nucleic Acid Strands: Enhanced Models, Robustness, and Speed”. In: (2019, in preparation).
*These authors contributed equally. NJP developed the recursions necessary to capture coaxial and dangle stacking states. MEF and NJP implemented the recursions. NJP developed and implemented the simultaneous sampling algorithm. NJP implemented the suboptimal structure generation algorithm. MEF developed and implemented the blockwise operation order, including the backtrack-free pair probability algorithm. MEF developed the evaluation algebra framework, and NJP and MEF programmed the evaluation algebras. NJP generated all designed sequences used in the benchmarks, and MEF ran analysis benchmarking calculations. NJP wrote the initial draft of the manuscript, and MEF, NJP, and NAP wrote the final manuscript and supplementary information.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

As a class of programmable molecular materials, nucleic acids have long interested both bioengineers and computer scientists with their promise of computing in solution. Cross-pollination between these fields produced the interdisciplinary subfield of molecular programming, which seeks to provide architectures and interfaces for embedding software in biology and chemistry. Molecular programs introduced to a cellular environment can modulate, co-opt, and integrate diverse cellular pathways. Forays in this area have produced actuators[1,2], logic gates (based on strand displacement[3] or cooperative hybridization[4]), signal transduction in biological contexts[5,6], and fluorescent-imaging signal amplification via hybridization chain reaction for use *in situ*[7–9]. These engineered systems rely on relatively small numbers of component strands (fewer than 10, and frequently fewer than 4) in their target reaction pathways, and each of these components is typically small (fewer than 100 nucleotides). Other work has pushed the size boundaries by looking at biologically-sized structures (hundreds to thousands of nucleotides), such as DNA origami[10] and cotranscriptionally-folded RNA origami[11], or constructing reaction pathways with large numbers of components (130 strands for a square-root gate based on the see-saw motif[12] or 168 toehold-switches[13]). With each past success spurring on increasingly ambitious projects, demand has increased for performant and accurate algorithms and software tools for analyzing and designing nucleic acid systems.

The development of algorithms for the analysis of thermodynamic properties of nucleic acids has spanned nearly four decades of research. We focus on algorithms using the nearest-neighbor energy model, which assigns free energies to a secondary structure by decomposing the structure into its component loops, each of which has a free energy functional form in the model (Chapter 2). The first of these algorithms used a dynamic programming algorithm to compute the minimum free energy (MFE) and MFE structure of a single strand of RNA[14]. Another dynamic programming algorithm, using a set of recursions that encounter each subsequence contribution uniquely, allowed computation of the partition function and base pairing probabilities of a single strand of RNA[15]. This type of dynamic program was extended to compute partition functions of complexes of two interacting strands, excluding intrastrand base pairs[16]. These partition functions could then be used to determine the equilibrium concentration of monomer and dimer species, assuming complexes of 3 or more strands do not form. The restriction to interstrand base pairs was lifted in later algorithm development[17]. The algorithms for computing MFE structures were then extended to complexes of two or more interacting strands[18]. Most recently, dynamic programming algorithms were developed to compute the partition function, base pairing probabilities, and MFE for the unpseudoknotted complex ensemble consisting of arbitrary numbers of interacting strands[19]. This was paired with a convex optimization algorithm to solve the coupled equilibria problem necessary to determine the partitioning of strands into complexes in a test tube ensemble. Together, this allows the analysis of concentration and base pairing properties of a test tube containing arbitrary concentrations of arbitrary types of strands forming arbitrary complexes[19].

The majority of work on nucleic acid design has focused on designing single nucleic acid complexes to adopt a target secondary structure[20–39]. These algorithms can be roughly classified by their energy models and objective functions. The first design algorithms developed involved the non-thermodynamic objective function of sequence symmetry minimization[20,21]. This objective function performs pure negative design by minimizing the number of off-target binding sites above a certain size within a complex. By far, the largest number of algorithms have focused on the MFE structure of a complex[22–26,28–33,35,36]. Generally, two possible objective functions are possible: minimization of the energy of the target structure, a form of positive design, and minimization of the base pair hamming distance from the MFE structure to the target structure, a form of negative design. There have been other complex design algorithms that used the thermodynamic energy model to consider the properties of the ensemble of sequences consistent with a given target structure rather than iteratively considering the structural ensemble of single sequences[27,34]. Finally the complex design algorithms from our lab[37–39] have focused on minimizing the ensemble defect, or average number of unpaired bases over the Boltzmann weighted ensemble of structures. This approach simultaneously implements a positive and negative design paradigm. Design efficiency is enhanced through the process of hierarchical ensemble decomposition, allowing the bulk of the optimization to happen on small segments of structures before evaluating the costly objective function in full[40]. This has been extended to the simultaneous design of multiple complexes which share sequence information in order to facilitate reaction pathway engineering[41]. Effective complex design ensures that given that individual strands associate into a complex, the target structure will be well-formed as determined by whatever objective function metric was used for design. Regardless of objective function, this is a necessary but insufficient condition to ensure that complex forms well in experimental dilute solution conditions. This is because the partitioning of strand species amongst complex species is determined by the relative free energies of the complexes, a scalar quantity which sums over the intracomplex secondary structure states[19]. As such, to ensure sequences form at the correct concentration in the correct target structure in a test tube ensemble, the test tube design algorithm[42] explicitly accounts for and designs against the formation of off-target complexes that, by conservation of mass, lead to deficiencies in on-target concentration while simultaneously reducing the complex ensemble defect. This allows minimization of the average concentration of incorrectly paired nucleotides over the ensemble of the target test tube, or test tube ensemble defect.

The work in this thesis continues these two lines of algorithm development. Chapter 2 introduces a new tripartite framework composed of recursions, evaluation algebras, and operations orders that unifies computation of thermodynamic quantities; the implementation allows for orders of magnitude increases in performance, computation over the complex and test tube ensembles including coaxial and dangle stacking states, and robustness to floating point overflow. In Chapter 3, we introduce an algorithm for simultaneous sequence design of multiple test tube ensembles through minimization of the multistate test tube ensemble defect subject to diverse user-specified hard sequence constraints in the service of reaction pathway engineering. Finally, the algorithm of Chapter 4 extends the algorithm of Chapter 3 by building on top of the framework for Chapter 2 and by adding soft constraints, which increase flexibility through new constraint types and improve performance for design of highly constrained reaction pathways. Taken together, the algorithms of Chapters 2 and 4 represent NUPACK 4.0, the next generation of software tools for the design and analysis of nucleic acid systems and reaction pathways.

Bibliography

[1] B. Yurke, A. J. Turberfield, A. P. Mills Jr, F. C. Simmel, and J. L. Neumann. "A DNA-fuelled molecular machine made of DNA". In: *Nature* 406.6796 (2000), p. 605.

[2] P. Yin, H. M. T. Choi, C. R. Calvert, and N. A. Pierce. "Programming biomolecular self-assembly pathways." In: *Nature* 451.7176 (2008), pp. 318–322. ISSN: 0028-0836. DOI: 10.1038/nature06451.

[3] G. Seelig, D. Soloveichik, D. Y. Zhang, and E. Winfree. "Enzyme-free nucleic acid logic circuits." In: *Science (New York, N.Y.)* 314.5805 (2006), pp. 1585–1588. ISSN: 0036-8075. DOI: 10.1126/science.1132493.

[4] D. Y. Zhang. "Cooperative hybridization of oligonucleotides". In: *Journal of the American Chemical Society* 133.4 (2011), pp. 1077–1086. ISSN: 00027863. DOI: 10.1021/ja109089q.

[5] L. M. Hochrein, M. Schwarzkopf, M. Shahgholi, P. Yin, and N. a. Pierce. "Conditional Dicer substrate formation via shape and sequence transduction with small conditional RNAs". In: *Journal of the American Chemical Society* 135.46 (2013), pp. 17322–17330. ISSN: 00027863. DOI: 10.1021/ja404676x.

[6] M. H. Hanewich-Hollatz, Z. Chen, L. M. Hochrein, J. Huang, and N. A. Pierce. "Conditional Guide RNAs: Programmable Conditional Regulation of CRISPR/Cas Function in Bacterial and Mammalian Cells via Dynamic RNA Nanotechnology". In: *ACS Central Science* (2019).

[7] R. M. Dirks and N. A. Pierce. "Triggered amplification by hybridization chain reaction." In: *Proceedings of the National Academy of Sciences of the United States of America* 101.43 (2004), pp. 15275–15278. ISSN: 0027-8424. DOI: 10.1073/pnas.0407024101.

[8] H. M. T. Choi, J. Y. Chang, L. a. Trinh, J. E. Padilla, S. E. Fraser, and N. A. Pierce. "Programmable in situ amplification for multiplexed imaging of mRNA expression." In: *Nature biotechnology* 28.11 (2010), pp. 1208–1212. ISSN: 1087-0156. DOI: 10.1038/nbt.1692.

[9] H. M. T. Choi, V. a. Beck, and N. A. Pierce. "Next-generation in situ hybridization chain reaction: Higher gain, lower cost, greater durability". In: *ACS Nano* 8.5 (2014), pp. 4284–4294. ISSN: 1936086X. DOI: 10.1021/nn405717p.

[10] P. W. Rothemund. "Folding DNA to create nanoscale shapes and patterns". In: *Nature* 440.7082 (2006), p. 297.

[11] C. Geary, P. W. K. Rothemund, and E. S. Andersen. "A single-stranded architecture for cotranscriptional folding of RNA nanostructures". In: *Science* 345.6198 (2014), pp. 799–804. ISSN: 0036-8075. DOI: 10.1126/science.1253920. URL: http://www.sciencemag.org/content/345/6198/799.short.

[12] L. Qian and E. Winfree. "Scaling up digital circuit computation with DNA strand displacement cascades". In: *Science* 332.June (2011), pp. 1196–1201.

[13] A. A. Green, P. A. Silver, J. J. Collins, and P. Yin. "Toehold switches: de-novo-designed regulators of gene expression". In: *Cell* 159.4 (2014), pp. 925–939.

[14] M. Zuker and P. Stiegler. “Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information”. In: *Nucleic Acids Research* 9.1 (1981), pp. 133–148. ISSN: 03051048. DOI: 10.1093/nar/9.1.133.

[15] J. McCaskill. “The equilibrium partition function and base pair binding probabilities for RNA secondary structure”. In: *Biopolymers* 29 (1990), pp. 1105–1119.

[16] R. Dimitrov and M. Zuker. “Prediction of hybridization and melting for double-stranded nucleic acids”. In: *Biophys. J.* 87.1 (2004), pp. 215–226.

[17] S. Bernhart, H. Tafer, U. Muckstein, C. Flamm, P. Stadler, and I. Hofacker. “Partition function and base pairing probabilities of RNA heterodimers”. In: *Algorithms Mol. Biol.* 1.3 (2006).

[18] M. Andronescu, Z. Zhang, and A. Condon. “Secondary structure prediction of interacting RNA molecules”. In: *J. Mol. Biol.* 345 (2005), pp. 987–1001.

[19] R. M. Dirks, J. S. Bois, J. M. Schaeffer, E. Winfree, and N. A. Pierce. “Thermodynamic Analysis of Interacting Nucleic Acid Strands”. In: *SIAM Rev.* 49.1 (2007), pp. 65–88.

[20] N. Seeman and N. Kallenbach. “Design of immobile nucleic acid junctions”. In: *Biophysical Journal* 44.2 (Nov. 1983), pp. 201–209. ISSN: 00063495. DOI: 10.1016/S0006-3495(83)84292-1. URL: https://linkinghub.elsevier.com/retrieve/pii/S0006349583842921.

[21] N. C. Seeman. “De novo design of sequences for nucleic acid structural engineering”. In: *Journal of Biomolecular Structure and Dynamics* 8.3 (1990), pp. 573–581. ISSN: 15380254. DOI: 10.1080/07391102.1990.10507829.

[22] I. Hofacker, W. Fontana, P. Stadler, L. Bonhoeffer, M. Tacker, and P. Schuster. “Fast folding and comparison of RNA secondary structures”. In: *Chem. Mon.* 125 (1994), pp. 167–188.

[23] C. Flamm, I. Hofacker, S. Maurer-Stroh, P. Stadler, and M. Zehl. “Design of multistable RNA molecules”. In: *RNA* 7 (2001), pp. 254–265.

[24] M. Andronescu, A. Fejes, F. Hutter, H. Hoos, and A. Condon. “A new algorithm for RNA secondary structure design”. In: *J. Mol. Biol.* 336.3 (2004), pp. 607–624.

[25] A. Busch and R. Backofen. “INFO-RNA–a fast approach to inverse RNA folding”. In: *Bioinformatics* 22.15 (2006), pp. 1823–1831.

[26] R. Aguirre-Hernández, H. Hoos, and A. Condon. “Computational RNA secondary structure design: empirical complexity and improved methods”. In: *BMC Bioinformatics* 8 (2007), Article 34.

[27] B. Burghardt and A. Hartmann. “RNA secondary structure design”. In: *Phys. Rev. E* 75 (2007), p. 021920.

[28] J. Z. M. Gao, L. Y. M. Li, and C. M. Reidys. “Inverse folding of RNA pseudoknot structures”. In: *Algorithms Mol. Biol.* 5 (2010), p. 27.

[29] W. J. Shu, M. Liu, H. B. Chen, X. C. Bo, and S. Q. Wang. “ARDesigner: A web-based system for allosteric RNA design”. In: *J. Biotechnol.* 150.4 (2010), pp. 466–473.

[30] A. Avihoo, A. Churkin, and D. Barash. “RNAexinv: An extended inverse RNA folding from shape and physical attributes to sequences”. In: *BMC Bioinformatics* 12 (2011), p. 319.

[31] E. I. Ramlan and K. P. Zauner. “Design of interacting multi-stable nucleic acids for molecular information processing”. In: *Biosystems* 105.1 (2011), pp. 14–24.

[32] A. Taneda. “MODENA: a multi-objective RNA inverse folding”. In: *Adv. Appl. Bioinforma. Chem.* 4 (2011), pp. 1–12.

[33] A. Levin, M. Lis, Y. Ponty, C. W. O’Donnell, S. Devadas, B. Berger, and J. Waldispühl. “A global sampling approach to designing and reengineering RNA secondary structures”. In: *Nucleic Acids Res.* 40.20 (2012), pp. 10041–10052.

[34] M. C. Matthies, S. Bienert, and A. E. Torda. “Dynamics in Sequence Space for RNA Secondary Structure Design”. In: *J. Chem. Theory Comput.* 8.10 (2012), pp. 3663–3670.

[35] A. Taneda. “Multi-objective genetic algorithm for pseudoknotted RNA sequence design”. In: *Front. Genet.* 3 (2012), p. 36.

[36] R. B. Lyngsø, J. W. J. Anderson, E. Sizikova, A. Badugu, T. Hyland, and J. Hein. “Frnakenstein: multiple target inverse RNA folding”. In: *BMC Bioinformatics* 13 (2012), p. 260.

[37] R. M. Dirks and N. A. Pierce. “A partition function algorithm for nucleic acid secondary structure including pseudoknots”. In: *Journal of Computational Chemistry* 24.13 (2003), pp. 1664–1677. ISSN: 01928651. DOI: 10.1002/jcc.10296.

[38] R. M. Dirks, M. Lin, E. Winfree, and N. A. Pierce. “Paradigms for computational nucleic acid design”. In: *Nucleic Acids Research* 32.4 (2004), pp. 1392–1403. ISSN: 03051048. DOI: 10.1093/nar/gkh291.

[39] J. N. Zadeh, B. R. Wolfe, and N. A. Pierce. “Nucleic acid sequence design via efficient ensemble defect optimization”. In: *Journal of Computational Chemistry* 32.3 (2011), pp. 439–452. DOI: 10.1002/jcc.21633. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jcc.21633. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.21633.

[40] J. N. Zadeh, C. D. Steenberg, J. S. Bois, B. R. Wolfe, M. B. Pierce, A. R. Khan, R. M. Dirks, and N. A. Pierce. “NUPACK: Analysis and design of nucleic acid systems”. In: *Journal of Computational Chemistry* 32.1 (Jan. 2011), pp. 170–173. ISSN: 01928651. DOI: 10.1002/jcc.21596. arXiv: NIHMS150003. URL: http://doi.wiley.com/10.1002/jcc.21596.

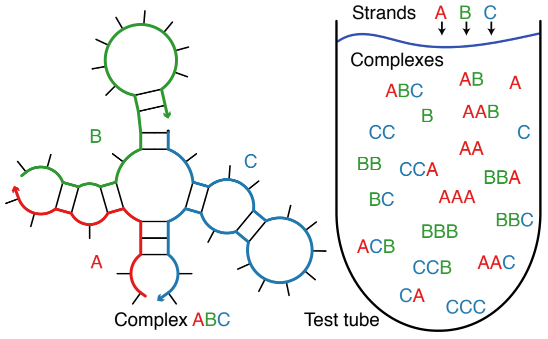
[41] J. N. Zadeh. “Algorithms for Nucleic Acid Sequence Design”. PhD thesis. 2010.

[42] B. R. Wolfe and N. A. Pierce. “Sequence Design for a Test Tube of Interacting Nucleic Acid Strands”. In: *ACS Synthetic Biology* (2014), p. 141020092749006. ISSN: 2161-5063. DOI: 10.1021/sb5002196. URL: http://pubs.acs.org/doi/pdfplus/10.1021/sb5002196.

Chapter 2

A Unified Dynamic Programming Framework for The Analysis of Interacting Nucleic Acid Strands: Enhanced Models, Robustness, and Speed

This chapter was adapted from material in M. E. Fornace*, N. J. Porubsky* and N. A. Pierce. "A Unified Dynamic Programming Framework for the Analysis of Interacting Nucleic Acid Strands: Enhanced Models, Robustness, and Speed". In: (2019, in preparation).



NUPACK algorithms enable analysis of nucleic acid sequences over complex and test tube ensembles containing arbitrary numbers of interacting strand species, serving the needs of researchers in molecular programming, nucleic acid nanotechnology, synthetic biology, and across the life sciences. Here, to enhance the underlying physical model, ensure robustness for large calculations, and achieve dramatic speedups when calculating diverse physical quantities over complex and test tube ensembles, we introduce a unified dynamic programming framework that combines three ingredients: 1) recursions that specify the dependencies between subproblems and incorporate the details of the structural ensemble and the free energy model, 2) evaluation algebras that define the mathematical form of each subproblem, 3) operation orders that specify the computational trajectory through the dependency graph of subproblems. The physical model is enhanced using new recursions that operate over the complex ensemble including coaxial and dangle stacking subensembles. The recursions are coded generically and then compiled with a quantity-specific evaluation algebra and operation order to generate an executable for each physical quantity: partition function, equilibrium pair probabilities, MFE energy, MFE structure(s), suboptimal structures, and Boltzmann sampled structures. For large complexes (e.g., 30,000 nt), robustness is achieved for partition function calculations using an overflow-safe evaluation algebra, and for equilibrium pair probabilities using a backtrack-free operation order. A new blockwise operation order that treats subcomplex blocks for the complex species in a test tube ensemble enables dramatic speedups (e.g., 20–120×) using vectorization and caching. With these performance enhancements, equilibrium analysis of substantial test tube ensembles can be performed in ≤ 1 minute on a single computational core (e.g., partition function and equilibrium concentration for all complex species of up to 6 strands formed from 2 strand species of 300 nt each, or for all complex species of up to 2 strands formed from 80 strand species of 100 nt each). A new sampling algorithm simultaneously samples multiple structures from the complex ensemble to yield speedups of an order of magnitude or more as the number of structures increases above $\approx 10^3$. These advances are available within the NUPACK 4.0 code base (`www.nupack.org`) which can be scripted using library calls to the all-new NUPACK Python module.

2.1 Introduction

NUPACK (Nucleic Acid Package) is a growing software suite for the analysis and design of nucleic acid structures, devices, and systems[1]. Algorithms are formulated in terms of nucleic acid secondary structure (i.e., the base pairs of a set of DNA or RNA strands), employing empirical free energy models[2–11]. NUPACK algorithms are unique in treating complex and test tube ensembles containing arbitrary numbers of interacting strand species, providing crucial tools for capturing concentration effects essential to analyzing and designing the intermolecular interactions that are a hallmark of molecular programming, nucleic acid nanotechnology, and synthetic biology.

Here, following 15 years of NUPACK algorithm development (NUPACK 1.0 to 3.2)[1,12–18], we reconsidered every algorithm, arriving at a new unified dynamic programming framework that leads to major improvements of five varieties:

- *Elucidation:* diverse physical quantities are calculated using dynamic programs each combining three ingredients: model-specific recursions, a quantity-specific evaluation algebra, and a quantity-specific operation order.
- *Model:* new recursions capture the structural and energetic details of coaxial and dangle stacking subensembles in the complex ensemble.
- *Robustness:* over-flow safe evaluation algebras and backtrack-free operation orders enable robust partition function and pair probability calculations for large complexes.
- *Speed:* new blockwise operation orders yield dramatic speedups of 1–2 orders of magnitude for equilibrium analysis of test tube ensembles.
- *Brevity:* use of a generic programming paradigm and compile-time polymorphism dramatically reduce the size of the code base.

We begin by defining the underlying physical model, including definitions of the complex and test tube structural ensembles, and specification of the free energy model for a complex ensemble including coaxial and dangle stacking subensembles. We then describe the unified dynamic programming framework, describing new recursions that capture the details of the enhanced physical model, and new evaluation algebras and operation orders that enable calculation of diverse physical quantities for complex and test tube ensembles of interacting DNA or RNA strands. The resulting suite of algorithms comprise the all-new NUPACK 4.0 analysis code base[19]. Enhanced models, robustness, and

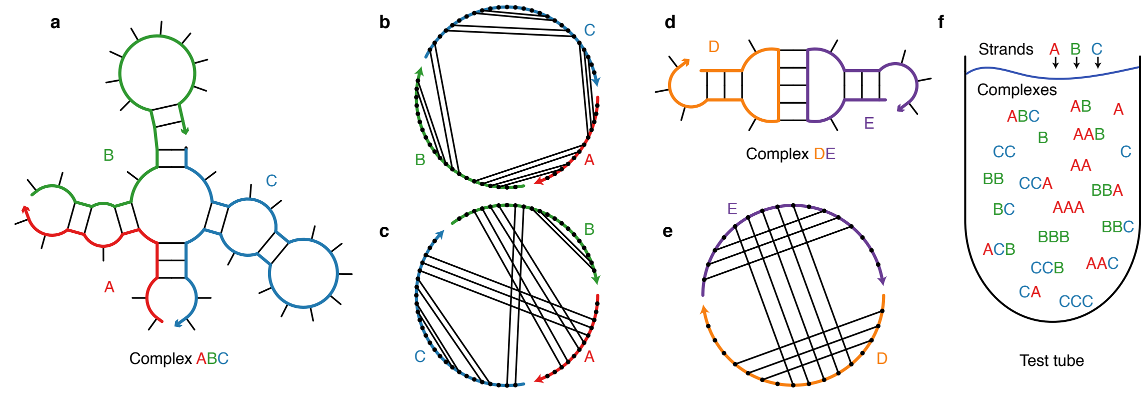


Figure 2.1: Complex and test tube ensembles. (a) A connected unpseudoknotted secondary structure for complex with strand ordering π = ABC. An arrowhead denotes the $3'$ end of each strand. (b) Polymer graph representation of the same secondary structure showing no crossing lines for strand ordering π = ABC. (c) Alternative strand ordering π = ACB yields a polymer graph with crossing lines. (c) A pseudoknotted secondary structure with base pairs $i \cdot j$ and $d \cdot e$ (with $i < d$) that fail to satisfy the nesting property $i < d < e < j$, yielding crossing lines in the corresponding polymer graph (e) for the sole strand ordering π = DE. (f) A test tube ensemble containing strand species Ψ^0 = {A,B,C} interacting to form all complex species Ψ of up to $L_{\max}$ = 3 strands.

speed will benefit researchers in molecular programming, nucleic acid nanotechnology, synthetic biology, and across the life sciences.

2.2 Physical Model

2.2.1 Complex Ensemble and Test Tube Ensembles

NUPACK algorithms operate over two fundamental ensembles:

- *Complex ensemble:* The ensemble of all (unpseudoknotted connected) secondary structures for an arbitrary number of interacting DNA or RNA strands.
- *Test tube ensemble:* The ensemble of a dilute solution containing an arbitrary number of DNA or RNA strand species (introduced at user-specified concentrations) interacting to form an arbitrary number of complex species.

Furthermore, to enable reaction pathway engineering of dynamic hybridization cascades (e.g., shape and sequence transduction using small conditional RNAs[20]) or large-scale structural engineering including pseudoknots (e.g., RNA origamis[21]), NUPACK generalizes sequence analysis and design to multi-complex and multi-tube ensembles[18].

The sequence, ϕ, of one or more interacting RNA strands is specified as a list of bases $\phi^a \in$ {`A`,`C`,`G`,`U`} for $a = 1, \ldots, |\phi|$. (Throughout our presentation, the situation for DNA is analogous to that for RNA, with `T` replacing `U`). A secondary structure, s, of one or more interacting RNA strands is defined by a set of base pairs (each a Watson–Crick pair [`A`·`U` or `C`·`G`] or a wobble pair [`G`·`U`]). For example, see the secondary structures of Figures 2.1ad.

For algorithmic purposes, it is convenient to describe secondary structures using a *polymer graph* representation, constructed by ordering the strands around a circle, drawing the backbones in succession from $5'$ to $3'$ around the circumference with a *nick* between each strand, and drawing straight lines connecting paired bases (Figure 2.2bc). A secondary structure is *unpseudoknotted* if there exists a strand ordering for which the polymer graph has no crossing lines (Figure 2.2b), or *pseudoknotted* if all strand orderings contain crossing lines (e.g., the kissing loops of Figure 2.2de). A secondary structure is *connected* if no subset of the strands is free of the others. Consider a *complex* of L distinct strands (e.g., each with a unique identifier in $\{1, \ldots, L\}$) corresponding to strand ordering π. The *complex* ensemble $\overline{\Gamma}$ contains all connected polymer graphs with no crossing lines for strand ordering π (i.e., all unpseudoknotted secondary structures)[15]. (We dispense with our prior convention[1,15,16] of calling this entity an *ordered complex*.)

As a matter of algorithmic necessity, all of the dynamic programs developed in the present work operate on complex ensemble $\overline{\Gamma}$ treating all strands as distinct. However, in the laboratory, strands with the same sequence are typically indistinguishable with respect to experimental observables. Hence, for comparison to experimental data, physical quantities calculated over ensemble $\overline{\Gamma}$ can be post-processed to obtain the corresponding quantities calculated over ensemble Γ in which strands with the same sequence are treated as indistinguishable (see Section A.7 for details). The ensemble $\Gamma \subseteq \overline{\Gamma}$ is a maximal subset of distinct secondary structures for strand ordering π. Two secondary structures are indistinguishable if their polymer graphs can be rotated so that all strands are mapped onto indistinguishable strands, all base pairs are mapped onto base pairs, and all unpaired bases are mapped onto unpaired bases; otherwise the structures are distinct[15].

A *test tube* ensemble is a dilute solution containing a set of strand species, Ψ^0, introduced at user-specified concentrations, that interact to form a set of complex species, Ψ, each corresponding to a different strand ordering treating strands with the same sequence as indistinguishable. For L strands, there are $(L-1)!$ strand orderings if all strands are different species (e.g., complexes ABC and ACB for $L = 3$ and strands A, B, C), but fewer than $(L-1)!$ strand orderings if some strands are of the same species (e.g., complex AAA for $L = 3$ for three A strands). By the Representation Theorem of Dirks et al.[15], a secondary structure in the complex ensemble for one strand ordering does not appear in the complex ensemble for any other strand ordering. It is often convenient to define Ψ to contain all complex species of up to $L_{\max}$ strands (e.g., Figure 2.2f), although Ψ can be defined to contain arbitrary complex species formed from strand species Ψ^0.

2.2.2 Loop-Based Free Energy Model

For each (unpseudoknotted) secondary structure $s \in \overline{\Gamma}$, the free energy, $\overline{\Delta G}(\phi, s)$, is estimated as the sum of the empirically determined free energies of the constituent loops[4–6,10,22,23] plus a strand association penalty[24], ΔG^{assoc}, applied $L-1$ times for a complex of L strands:

$$\overline{\Delta G}(\phi, s) = (L-1)\,\Delta G^{\text{assoc}} + \sum_{\text{loop} \in s} \Delta G(\text{loop}). \tag{2.1}$$

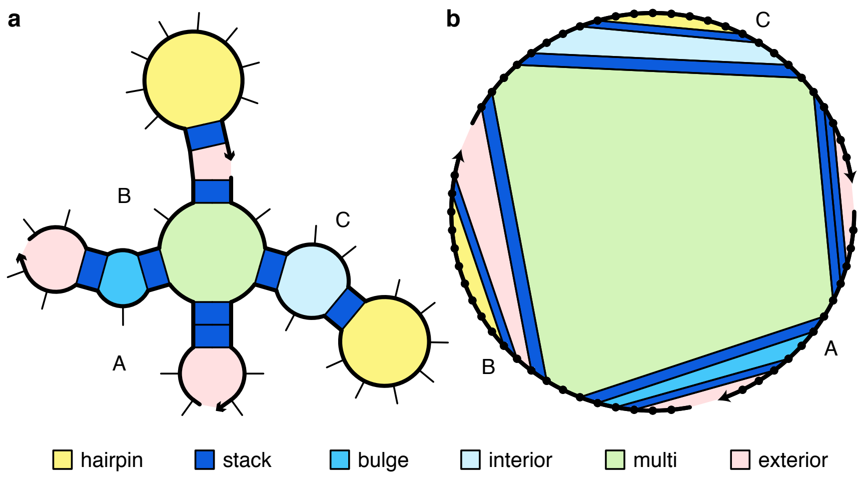


Figure 2.2: Loop-based free energy model for a complex. (a) Canonical loop types for complex with strand ordering π = ABC. (b) Equivalent polymer graph representation. An arrowhead denotes the 3′ end of each strand.

The secondary structure and polymer graph of Figure 2.2 illustrate the different loop types, with free energies modeled as follows[4–6,10,22,25]:

- A *hairpin loop* is closed by a single base-pair $i \cdot j$. The loop free energy, $\Delta G^{\text{hairpin}}_{i,j}$, depends on sequence and loop size.
- An *interior loop* is closed by two base pairs ($i \cdot j$ and $d \cdot e$ with $i < d < e < j$). The loop free energy, $\Delta G^{\text{interior}}_{i,d,e,j}$ depends on sequence, loop size, and loop asymmetry. *Bulge loops* (where either $d = i + 1$ or $e = j - 1$) and *stacked pairs* (where both $d = i + 1$ and $e = j - 1$) are treated as special cases of interior loops.
- A *multiloop* is closed by three or more base pairs. The loop free energy is modeled as the sum of three sequence-independent penalties: (1) $\Delta G^{\text{multi}}_{\text{init}}$ for formation of a multiloop, (2) $\Delta G^{\text{multi}}_{\text{bp}}$ for each closing base pair, (3) $\Delta G^{\text{multi}}_{\text{nt}}$ for each unpaired nucleotide inside the multiloop, plus a sequence-dependent penalty: (4) $\Delta G^{\text{terminal}}_{i,j}$ for each closing pair $i \cdot j$.
- An *exterior loop* contains a strand break and any number of closing base pairs. The exterior loop free energy is the sum of $\Delta G^{\text{terminal}}_{i,j}$ over all closing base pairs $i \cdot j$, so an unpaired strand has a free energy of zero[15].

2.2.3 Coaxial and Dangle Stacking Subensembles within Complex Ensembles

Within a multiloop or an exterior loop, there is a subensemble of coaxial stacking states between adjacent closing base pairs and dangle stacking states between closing base pairs and adjacent unpaired bases. The physical model for multiloops and exterior loops has previously been enhanced for the ensemble of a single strand[8] by incorporating coaxial stacking[5,25,26] and dangle stacking[3,6,25,27] terms into the multiloop and exterior loop free energies. For the complex ensemble, we have

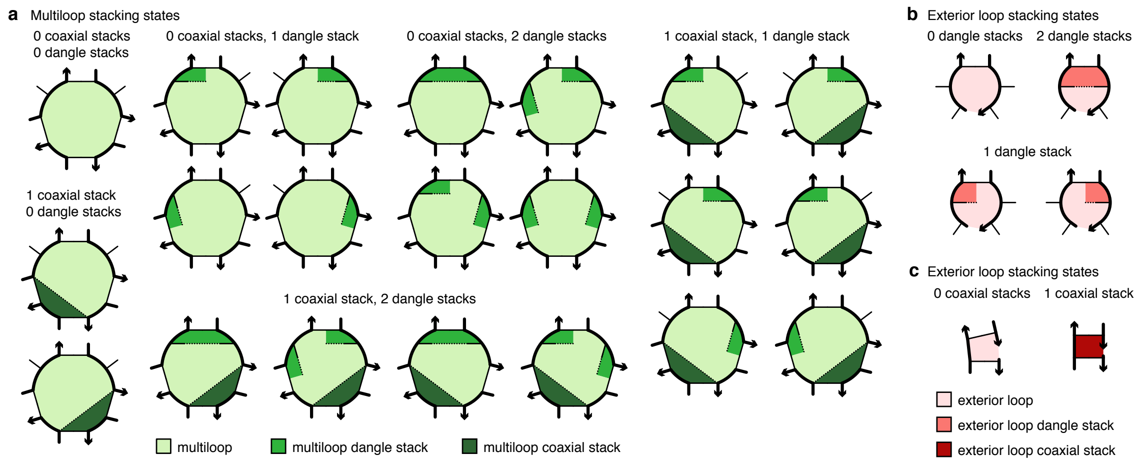

a Multiloop stacking states
0 coaxial stacks 0 dangle stacks
0 coaxial stacks, 1 dangle stack
0 coaxial stacks, 2 dangle stacks
1 coaxial stack, 1 dangle stack
1 coaxial stack 0 dangle stacks
1 coaxial stack, 2 dangle stacks
multiloop
multiloop dangle stack
multiloop coaxial stack
b Exterior loop stacking states
0 dangle stacks
2 dangle stacks
1 dangle stack
c Exterior loop stacking states
0 coaxial stacks
1 coaxial stack
exterior loop
exterior loop dangle stack
exterior loop coaxial stack

Figure 2.3: Coaxial and dangle stacking states for multiloops and exterior loops. (a) Stacking subensemble for the multiloop of Figure 2.2a. (b,c) Stacking subensembes for two external loops from Figure 2.2a.

previously neglected coaxial stacking and incorporated a heuristic dangle stacking state[15]. Here, we exactly incorporate all coaxial and dangle stacking states in the complex ensemble. Within a multiloop or exterior loop, a base pair can form one *coaxial stack* with an adjacent base pair, or can form a *dangle stack* with at most two adjacent unpaired bases; unpaired bases can either form no stack, or can form a dangle stack with at most one adjacent base pair. See Figure 2.3 for an illustration of the valid stacking states for a multiloop (panel a) or two exterior loops (panels b and c).

For a given multiloop or exterior loop, the energetic contributions of all possible coaxial and dangle stacking states are enumerated so as to calculate the free energy:

$$\Delta G^{\text{stacking}} = -kT \log \sum_{\omega \in \text{loop}} \prod_{x \in \omega} e^{-\Delta G_x / kT} \tag{2.2}$$

where ω indexes the possible stacking states within the loop and x indexes the individual stacks (coaxial or dangle) within a stacking state. The free energy of a multiloop or exterior loop is augmented by the corresponding $\Delta G^{\text{stacking}}$ bonus. Hence, a secondary structure s continues to be defined as a set of base pairs, and the stacking states within a given multiloop or exterior loop are treated as a structural subensemble that contributes in a Boltzmann-weighted fashion to the free energy model for the loop. Let $s^{\shortparallel} \in s$ denote a stacking state of the paired and unpaired bases in s. We may equivalently define the free energy of secondary structure s in terms of the free energies for all stacking states $s^{\shortparallel} \in s$:

$$\overline{\Delta G}(\phi, s) = -kT \log \sum_{s^{\shortparallel} \in s} e^{-\overline{\Delta G}(\phi, s^{\shortparallel})/kT} \tag{2.3}$$

Let $\overline{\Gamma}^{\shortparallel}$ denote the ensemble of stacking states corresponding to the complex ensemble of secondary structures $\overline{\Gamma}$.

2.2.4 Symmetry Correction

For a secondary structure $s \in \Gamma$ with an R-fold rotational symmetry there is in R-fold reduction in distinguishable conformational space, so the free energy (2.1) must be adjusted[15] by a symmetry correction:

$$\Delta G(\phi, s) = \overline{\Delta G}(\phi, s) + \Delta G^{\text{sym}}(\phi, s). \tag{2.4}$$

where

$$\Delta G^{\text{sym}}(\phi, s) = kT \log R(\phi, s). \tag{2.5}$$

Because the symmetry factor $R(\phi, s)$ is a global property of each secondary structure $s \in \Gamma$, it is not suitable for use with dynamic programs that treat multiple subproblems simultaneously without access to global structural information. As a result, dynamic programs operate on ensemble $\overline{\Gamma}$ using physical model (2.1) and then the Distinguishability Correction Theorem of Dirks et al.[15] enables exact conversion of physical quantities to ensemble Γ using physical model (2.4). Interestingly, ensembles $\overline{\Gamma}$ and Γ both have utility when examining the physical properties of a complex as they provide related but different perspectives, akin to complementary thought experiments (see Section A.7).

2.2.5 Free Energy Parameters

For RNA, we employ temperature-dependent parameters[5,6,10,22,25] including coaxial[5,25] and dangle[3,6,25] parameters in 1M Na$^+$. For DNA, we employ temperature-dependent parameters[4,6] including coaxial[26] and dangle[6,27] parameters in user-specified concentrations of Na$^+$ and Mg^{++}[4,7,26] (see Section A.1.2 for details on implementation of the salt corrections).

2.3 Algorithms

2.3.1 Physical Quantities

Consider a complex with sequence ϕ. We provide dynamic programs to calculate:

- the partition function,

$$\overline{Q}(\phi) = \sum_{s \in \overline{\Gamma}} e^{-\overline{\Delta G}(\phi, s)/kT}, \tag{2.6}$$

  over ensemble $\overline{\Gamma}$ treating all strands as distinct. Post-processing then yields the partition function $Q(\phi)$ over ensemble Γ treating strands with the same sequence as indistinguishable[15]. The equilibrium probability of any secondary structure $s \in \Gamma$ is then

$$p(\phi, s) = e^{-\Delta G(\phi, s)/kT} / Q(\phi). \tag{2.7}$$

- the base-pairing probability matrix $P(\phi)$ with entries $P^{i,j}(\phi) \in [0, 1]$ corresponding to the probability

$$P^{i,j}(\phi) = \sum_{s \in \overline{\Gamma}} p(\phi, s) S^{i,j}(s) \tag{2.8}$$

that base pair $i \cdot j$ forms at equilibrium within ensemble $\overline{\Gamma}$, treating all strands as distinct. Here, $S(s)$ is a *structure matrix* with entries $S^{i,j}(s) = 1$ if structure s contains base pair $i \cdot j$ and $S^{i,j}(s) = 0$ otherwise. Abusing notation, the entry $S^{i,i}(s)$ is 1 if base i is unpaired in structure s and 0 otherwise; the entry $P^{i,i}(\phi) \in [0, 1]$ denotes the equilibrium probability that base i is unpaired over ensemble $\overline{\Gamma}$. Hence $S(s)$ and $P(\phi)$ are symmetric matrices with row and column sums of 1.

- the free energy of the minimum free energy (MFE) stacking state $s^{\shortparallel}_{\text{MFE}}(\phi) \in \overline{\Gamma}^{\shortparallel}$ treating all strands as distinct:

$$\overline{\Delta G}(\phi, s^{\shortparallel}_{\text{MFE}}) = \min_{s^{\shortparallel} \in \overline{\Gamma}^{\shortparallel}} \overline{\Delta G}(\phi, s^{\shortparallel}). \tag{2.9}$$

- the MFE proxy structure

$$s_{\text{MFE}'} = \{s \in \overline{\Gamma} | s^{\shortparallel}_{\text{MFE}} \in s, s^{\shortparallel}_{\text{MFE}}(\phi) = \arg\min_{s^{\shortparallel} \in \overline{\Gamma}^{\shortparallel}} \overline{\Delta G}(\phi, s^{\shortparallel})\}. \tag{2.10}$$

  defined as the secondary structure containing the MFE stacking state within its subensemble. If there is more than one MFE stacking state, the algorithm returns all corresponding MFE proxy structures.

- the set of suboptimal secondary structures

$$\begin{aligned} &\overline{\Gamma}_{\text{subopt}}(\phi, \Delta G_{\text{gap}}) = \\ &\qquad \{s \in \overline{\Gamma} | s^{\shortparallel} \in s, \overline{\Delta G}(\phi, s^{\shortparallel}) \leq \overline{\Delta G}(\phi, s^{\shortparallel}_{\text{MFE}}) + \Delta G_{\text{gap}}\} \end{aligned} \tag{2.11}$$

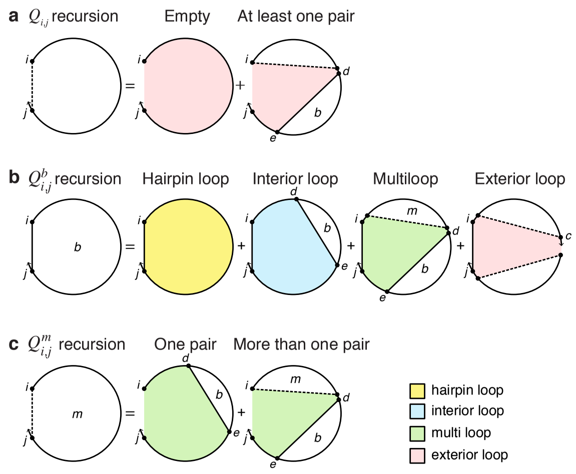
  with stacking states within a specified $\Delta G_{\text{gap}} \geq 0$ of the MFE stacking state.

- a set of J secondary structures Boltzmann sampled from ensemble $\overline{\Gamma}$ treating all strands as distinct:

$$\overline{\Gamma}_{\text{sample}}(\phi, J) \in \overline{\Gamma} \tag{2.12}$$

  Post-processing then yields the set of J secondary structures Boltzmann sampled from ensemble Γ treating strands with the same sequence as indistinguishable:

$$\Gamma_{\text{sample}}(\phi, J) \in \Gamma. \tag{2.13}$$



$$Q_{i,j} = 1 + \sum_{i \le d < e \le j} Q_{i,d-1}\, Q^b_{d,e}$$

$$\begin{aligned} Q^b_{i,j} = {} & \exp\{-\Delta G^{\text{hairpin}}_{i,j}/kT\} + \sum_{i<d<e<j} Q^b_{d,e} \exp\{-\Delta G^{\text{interior}}_{i,d,e,j}/kT\} \\ & + \sum_{i<d<e<j} Q^m_{i+1,d-1} Q^b_{d,e} \exp\{-[\Delta G^{\text{multi}}_{\text{init}} + 2\Delta G^{\text{multi}}_{\text{bp}} + n_{e+1,j-1}\Delta G^{\text{multi}}_{\text{nt}}]/kT\} \\ & + \sum_{\substack{i \le c < j \\ \text{s.t. } c+\frac{1}{2} \text{ is a nick}}} Q_{i+1,c}\, Q_{c+1,j-1} \end{aligned}$$

$$\begin{aligned} Q^m_{i,j} = {} & \sum_{i \le d < e \le j} Q^b_{d,e} \exp\{-[\Delta G^{\text{multi}}_{\text{bp}} + (n_{i,d-1} + n_{e+1,j})\Delta G^{\text{multi}}_{\text{nt}}]/kT\} \\ & + \sum_{i \le d < e \le j} Q^m_{i,d-1}\, Q^b_{d,e} \exp\{-[\Delta G^{\text{multi}}_{\text{bp}} + n_{e+1,j}\Delta G^{\text{multi}}_{\text{nt}}]/kT\} \end{aligned}$$

Figure 2.4: Partition function dynamic program recursion diagrams (left) and recursion equations (right)[15]. A solid straight line indicates a base pair and a dashed line demarcates a region without implying that the connected bases are paired. Shaded regions correspond to loop free energies that are explicitly incorporated at the current level of recursion (colors correspond to the loop types of Figure 2.2). (a) $Q_{i,j}$ represents the partition function for subsequence $[i, j]$. There are two cases: either there are no base pairs (corresponding to the reference state 0 and partition function contribution 1) or there is a $3'$-most base pair $d \cdot e$. In the latter case, determination of the partition function contribution makes use of previously computed subsequence partition functions $Q^b_{d,e}$ and $Q_{i,d-1}$. By the distributive law, multiplication of these subsequence partition functions (each representing a sum over substructures) implicitly sums over all pairwise combinations of substructures. The independence of the loop contributions in the energy model (2.1) implies that these products appropriately add the free energies in the exponents. (b) $Q^b_{i,j}$ is the partition function for subsequence $[i, j]$ with the restriction that bases i and j are paired. There are four cases: either there are no additional base pairs (corresponding to a hairpin loop), there is exactly one additional base pair $d \cdot e$ (corresponding to an interior loop), there is more than one additional base pair (corresponding to a multiloop) with $3'$-most pair $d \cdot e$ and at least one additional pair specified in a previously computed subsequence partition function $Q^m_{i,d-1}$, or there is an exterior loop containing a nick at $c + \frac{1}{2}$. $n_{i,j} \equiv j - i + 1$ denotes the number of nucleotides between i and j inclusive. (c) $Q^m_{i,j}$ is the partition function for subsequence $[i, j]$ with the restrictions that the subsequence is inside a multiloop and contains at least one base pair. There are two cases: either there is exactly one additional base pair $d \cdot e$ defining the multiloop, or there is more than one additional base pair defining the multiloop (with $3'$-most pair $d \cdot e$). The time complexity is $O(N^4)$ (indices i, d, e, j on the right hand side) and the space complexity is $O(N^2)$ (indices i, j on the left hand side).

Now consider a test tube ensemble containing an arbitrary set of strand species Ψ^0 interacting to form an arbitrary set of complex species Ψ. We provide algorithms to calculate:

- the set of equilibrium concentrations $x_\Psi \equiv x_c \ \ \forall c \in \Psi$, (specified as mole fractions) that are the unique solution to the strictly convex optimization problem[15]:

$$\min_{x_\Psi} \sum_{c \in \Psi} x_c(\log x_c - \log Q_c - 1) \tag{2.14a}$$

$$\text{subject to} \quad \sum_{c \in \Psi} A_{i,c} x_c = x_i^0 \quad \forall i \in \Psi^0, \tag{2.14b}$$

  expressed in terms of the previously calculated set of partition functions Q_Ψ. The constraints impose conservation of mass: A is the stoichiometry matrix such that $A_{i,c}$ is the number of strands of type i in complex c, and x_i^0 is the total concentration of strand i present in the test tube. Based on dimensional analysis[15], the algorithm operates on mole fractions, but for convenience, accepts molar strand concentrations $[i]^0 = x_i^0 \rho_{\mathrm{H_2O}}$ as inputs and returns molar complex concentrations $[c] = x_c \rho_{\mathrm{H_2O}}$ as outputs, where $\rho_{\mathrm{H_2O}}$ is the molarity of water.

- the ensemble pair fractions for the test tube ensemble, for example

$$f_A(i_A \cdot j_B) \tag{2.15}$$

  denotes the fraction of A strands that form base pair $i_A \cdot j_B$ (correspondingly $f_B(i_A \cdot j_B)$ fraction of B strands with base pair $i_A \cdot j_B$). In order to calculate these base-pairing observables, it is first necessary to calculate the set of equilibrium concentrations x_Ψ and the set of base-pairing probability matrices P_Ψ.

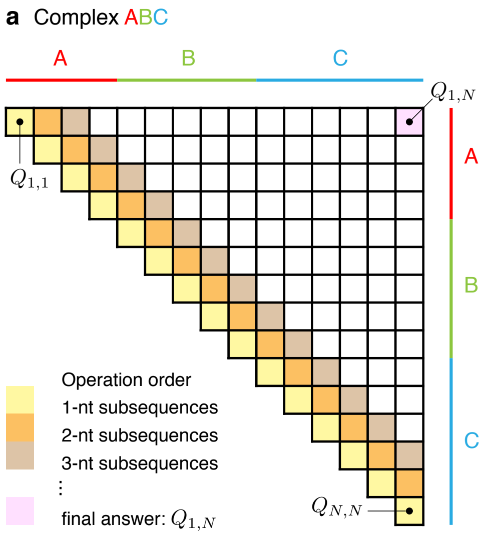


Figure 2.5: Operation order for partition function dynamic program over a complex ensemble with N nucleotides.

Table 2.1: Algorithmic ingredients for calculating diverse physical quantities.

Quantity	Symbol	Recursions	Evaluation Algebra	Dependency	Operation Order
Partition function	$\overline{Q}(\phi)$	Stacking	PFUNC, OVERFLOW	–	Blockwise forward sweep
MFE	$\overline{\Delta G}(\phi, s^{''}_{\mathrm{MFE}})$	Stacking	MFE	–	Blockwise forward sweep
Complex ensemble size	$\lvert\overline{\Gamma}\rvert$	No stacking	COUNT	–	Blockwise forward sweep
Pair probability matrix	$P(\phi)$	Stacking	PFUNC, OVERFLOW	–	Blockwise forward sweep
MFE structure proxy	$s_{\mathrm{MFE}'}(\phi)$	Stacking	MFESTRUC	$\overline{\Delta G}(\phi, s^{''}_{\mathrm{MFE}})$	Backtracking, stack
Suboptimal ensemble	$\overline{\Gamma}_{\mathrm{subopt}}(\phi, \Delta G_{\mathrm{gap}})$	Stacking	SUBOPT	$\overline{\Delta G}(\phi, s^{''}_{\mathrm{MFE}})$	Backtracking, stack
Sampled ensemble	$\overline{\Gamma}_{\mathrm{sample}}(\phi, J)$	Stacking	SAMPLE	$\overline{Q}(\phi)$	Backtracking, priority queue
Concentrations	x_{Ψ}	–	–	Q_{Ψ}	Convex optimization
Ensemble pair fractions	$f_A(i_A \cdot j_B)$	–	–	x_{Ψ}, P_{Ψ}	–

2.3.2 Existing Dynamic Programs

Before describing the new unified dynamic programming framework, it is helpful to briefly summarize existing algorithms that operate on complex ensemble $\overline{\Gamma}$ using a simplified free energy model that neglects coaxial stacking and approximates dangle stacking[15]. The complex ensemble size, $|\overline{\Gamma}|$, grows exponentially with the number of nucleotides (Figure A.34), $N \equiv |\phi|$, but the partition function can be calculated in $O(N^3)$ time and $O(N^2)$ space using a dynamic program[15,28]. The algorithm calculates the subsequence partition function $Q_{i,j}$ for each subsequence $[i, j]$ via a forward sweep from short subsequences to the full sequence (Figure 2.5), finally yielding the partition function of the full sequence, $Q_{1,N}$. The recursions used to calculate $Q_{i,j}$ from previously calculated subsequence partition functions can be depicted as recursion diagrams (Figure 2.4 left; with free energy contributions colored to match the loop types of Figure 2.2) or equivalently using recursion equations (Figure 2.4 right). The Q recursion relies on additional restricted partition functions Q^b and Q^m that are also calculated recursively. Collectively, the Q, Q^b, and Q^m recursions yield $\overline{Q}(\phi) = Q_{1,N}$, incorporating the partition function contributions of every structure $s \in \overline{\Gamma}$ based on free energy model (2.1) treating all strands as distinct. The Distinguishability Correction Theorem of Dirks et al.[15] then enables straightforward calculation of the partition function, $Q(\phi)$, over ensemble Γ using free energy model (2.4), treating strands with the same sequence as indistinguishable (see Section A.7). After calculating the partition function with a forward sweep from short to long sequences, dynamic programs that backtrack through the matrix of subsequence partition functions from long to short subsequences can be used to calculate the matrix of equilibrium base-pairing probabilities, $P(\phi)$,[14,15,28] or to Boltzmann sample a structure from ensemble $\overline{\Gamma}$[15,29].

The partition function dynamic program can be converted into an MFE dynamic program in a straightforward way by replacing every product of exponentiated free energies with a sum of free energies and every sum of alternative partition function contributions with a minimization over alternative free energy contributions, yielding the MFE of the full sequence, $\overline{\Delta G}(\phi, s_{\mathrm{MFE}}) = F_{1,N}$[15,30]. After calculating the MFE with a forward sweep from short to long subsequences, dynamic programs that backtrack through the matrix of subsequence MFEs from long to short subsequences can be used to determine the MFE secondary structure(s), $s_{\mathrm{MFE}}(\phi) \in \overline{\Gamma}$, or the ensemble of suboptimal structures, $\overline{\Gamma}_{\mathrm{subopt}}(\phi, \Delta G^{\mathrm{gap}})$. At the heart of the improvements in the present work is a new unified treatment of this suite of dynamic programs for calculating diverse physical quantities.

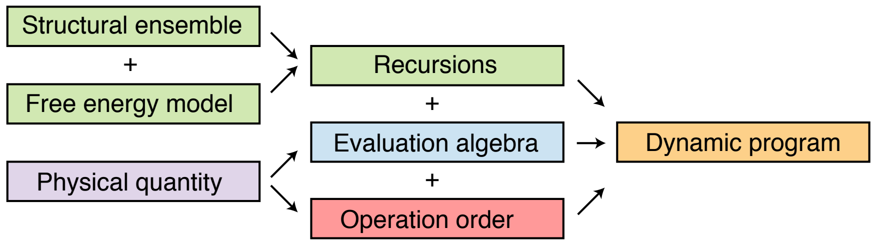
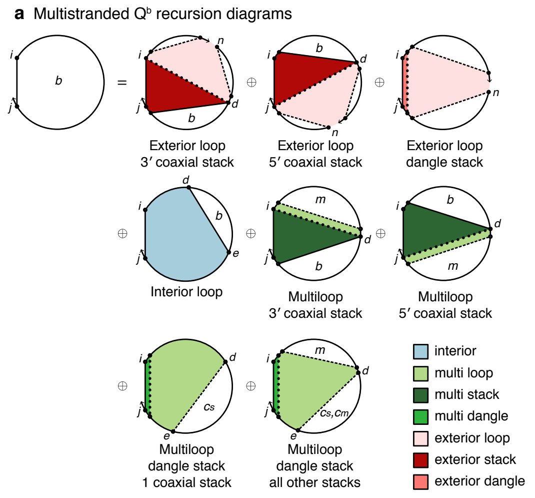


Figure 2.6: Unified dynamic programming framework. To calculate a physical quantity of interest based on a physical model comprising a structural ensemble and a free energy model, each dynamic program combines three ingredients: model-specific recursions, a quantity-specific evaluation algebra, and a quantity-specific operation order.

2.3.3 Unified Dynamic Programming Framework

In the new unified framework, each dynamic program combines three ingredients (Figure 2.6): a set of recursions, an evaluation algebra, and an operation order. A set of recursions specifies the dependencies of each subproblem, capturing the structural details of the complex ensemble and the energetic details of the loop-based free energy model. An evaluation algebra yields the mathematical form of each subproblem, allowing recursions to be generically extended to each physical quantity of interest. An operation order defines the computational trajectory through the dependency graph of subproblems, yielding dramatic speedups using appropriate data structures. In the following sections, we first introduce a new set of recursions that treat the enhanced physical model including coaxial and dangle stacking, and then describe evaluation algebras and operation orders that enable calculation of diverse physical quantities for complex and test tube ensembles (Table 2.1).



b Multistranded Qᵇ recursion equations for exterior loop terms

$$\bigoplus_{\substack{i+2<d<j\\ d\notin\eta}} \bigoplus_{\substack{i+1<n<d\\ n\in\eta}} W(\Delta G^{\text{coax}}_{d,j-1,i}(\phi)) \otimes Q_{i+1,n-1}Q_{n,d-1} \otimes Q^b_{d,j-1}$$

Exterior loop 3′ coaxial stack

$$\oplus \bigoplus_{\substack{i+1<d<j-1\\ (d+1)\notin\eta}} \bigoplus_{\substack{d+1<n<j\\ n\in\eta}} W(\Delta G^{\text{coax}}_{j,i,d}(\phi)) \otimes Q^b_{i+1,d} \otimes Q_{d+1,n-1}Q_{n,j-1}$$

Exterior loop 5′ coaxial stack

$$\oplus \bigoplus_{k,l\in\{0,1\}} \bigoplus_{\substack{i+k+1<n<j-l\\ n\in\eta}} W(\Delta G^{\text{dangle}}_{j-l,j,i,i+k}) \otimes Q_{i+k+1,n-1} \otimes Q_{n,j-l-1}$$

Exterior loop dangle stack

Figure 2.7: Representative recursion diagrams and equations operating over coaxial and dangle stacking subensembles within the complex ensemble. (a) Q^b recursion diagrams (cf. Q^b recursion diagrams of Figure 2.4b for the case without coaxial and dangle stacking). (b) Recursion equations corresponding to the exterior loop diagrams in the top row of panel (a) (cf. the exterior loop term in the Q^b recursion equation of Figure 2.4b for the case without coaxial and dangle stacking). Recursion equations use evaluation algebra operators $\oplus$ and $\otimes$, which for the partition function denote standard addition and multiplication (see Table 2.2 for evaluation algebras corresponding to other physical quantities).

	Algebra	Algorithm Output	$\mathbb{0}$	$\mathbb{1}$	$a \oplus b$	$a \otimes b$	$W(g)$
a	SumProduct	Partition function	0	1	$a + b$	$a \cdot b$	$\exp\left(\frac{-g}{k_B T}\right)$
	StructureCount	Ensemble size	0	1	$a + b$	$a \cdot b$	1
	MinSum	MFE	∞	0	$\min(a, b)$	$a + b$	g
b	SplitExponent	Partition function					
	Mantissa		0	1	$a_\mathrm{m} \cdot 2^{a_\mathrm{e}+\gamma} + b_\mathrm{m} \cdot 2^{b_\mathrm{e}+\gamma}$	$a_\mathrm{m} \cdot b_\mathrm{m}$	$\exp\left(\frac{-g}{k_B T}\right)$
	Exponent		0	γ	0	$a_\mathrm{e} + b_\mathrm{e} + \gamma$	γ
c	ArgRandom	Sampled structure					
	Value		0	1	$a_\mathrm{v} + b_\mathrm{v}$	$a_\mathrm{v} \cdot b_\mathrm{v}$	$\exp\left(\frac{-g}{k_B T}\right)$
	Elements		$\varnothing$	$\varnothing$	$\arg\text{random}(a_\mathrm{v}, b_\mathrm{v})$	$a_\lambda \cup b_\lambda$	$\varnothing$
d	ArgMin	MFE structure					
	Value		∞	0	$\min(a_\mathrm{v}, b_\mathrm{v})$	$a_\mathrm{v} + b_\mathrm{v}$	g
	Elements		$\varnothing$	$\varnothing$	$\arg\min(a_\mathrm{v}, b_\mathrm{v})$	$a_\lambda \cup b_\lambda$	$\varnothing$

Table 2.2: Evaluation algebras for dynamic programming algorithms. a and b are elements within the evaluation algebra domain. SumProduct yields the partition function of the complex ensemble. StructureCount yields the number of secondary structures in the complex ensemble. MinSum yields the free energy of the minimum free energy (MFE) stacking state in the complex ensemble. SplitExponent yields the partition function in split mantissa/exponent form using a given exponent shift γ in order to avoid overflow. ArgRandom yields a Boltzmann sampled structure associated with recursion elements x_λ from the complex ensemble with partition function x_v. ArgMin yields the secondary structure associated with recursion elements x_λ which contains the MFE stacking state with free energy x_v. See Section A.5 for details.

2.3.4 Recursions for the Complex Ensemble with Coaxial and Dangle Stacking

To treat the enhanced physical model including coaxial and dangle stacking contributions for all multiloops and exterior loops, we require a new set of recursions that incorporate the subensemble of stacking states and free energies defined by equation (2.2) and illustrated in Figure 2.3. To illustrate the nature of the changes to the recursion diagrams, Figure 2.7a displays $O(N^4)$ recursions for calculating interstrand contributions to $Q^b_{i,j}$ (the partition function for subsequence $[i, j]$ with i paired to j) including coaxial and dangle stacking; recursion energies incorporated by a given recursion are shaded according to the loop types of Figure 2.2 and stacking types of Figure 2.3. The full set of $O(N^4)$ recursions and more efficient $O(N^3)$ variants are provided in Section A.4. In the following sections, we describe how diverse physical quantities can be calculated using these recursions in combination with different evaluation algebras and operation orders.

2.3.5 Evaluation Algebras for Partition Function, Minimum Free Energy, and Ensemble Size

As previously noted for the complex ensemble without coaxial and dangle stacking, the partition function recursion diagrams of Figure 2.4a can alternatively be expressed as the partition function recursion equations of Figure 2.4b, and these in turn can be systematically transformed into recursion equations to calculate the MFE. Alternatively, we may view the partition function and MFE recursion equations as the results of applying two different evaluation algebras to a generic set of recursion diagrams and equations that capture the details of a given physical model (comprising a structural ensemble and a free energy model). Here, we formalize an *evaluation algebra* as an algebraic structure composed of (1) a semiring R equipped with commutative binary operators $\oplus$ and $\otimes$ and associated identity elements $\mathbb{0}$ and $\mathbb{1}$, (2) a map W from free energy parameters to R, and (3) a map Q from recursion indices to R. Table 2.2a defines the evaluation algebras for the partition function and MFE algorithms, as well as the evaluation algebra for calculating the size of the complex ensemble, $|\bar{\Gamma}|$. For example, for the partition function, (1) $\oplus$ is standard addition, $\otimes$ is standard multiplication, $\mathbb{0}$ is 0, $\mathbb{1}$ is 1, (2) $W(g)$ is the Boltzmann factor $\exp(-\Delta G/kT)$, and (3) Q is the trivial matrix lookup operator $Q(n,i,j) \mapsto Q^n_{i,j}$, where n denotes the type of recursion (e.g., $n = b$ for a Q^b recursion). The evaluation algebras for the partition function, MFE, and ensemble size can be applied to recursions that operate over the complex ensemble without or with coaxial and dangle stacking subensembles.

This paradigm of applying a quantity-specific evaluation algebra to a model-specific set of recursions extends to diverse physical quantities, as we describe in the sections that follow. This generic programming abstraction dramatically reduces the size of the code base and enforces implementation correctness. Instead of writing separate code to upgrade the recursion equations to the new physical model for each physical quantity, a single set of recursion equations is coded and compiled using C++ expression templates for each of the evaluation algebras in Table 2.2 to produce a suite of executables for calculating the corresponding physical quantities.

2.3.6 Overflow-Safe Evaluation Algebra for Large Partition Function Calculations.

One of the challenges with calculating the partition function is the prevention of overflow as the size of the complex, $N \equiv |\phi|$, increases. Using double-precision (64-bit) arithmetic, the maximum expressible number is $\approx 10^{308}$, enabling calculation of partition functions for complexes of ≈1400 nt for random sequences and ≈450 nt for designed sequences (with a deep well on the free energy landscape). Using quadruple-precision (128-bit) arithmetic, the maximum expressible number is increased to $\approx 10^{4932}$ (platform-dependent), which enables partition function calculations for complexes of up to ≈22,000 nt for random sequences and ≈7000 nt for designed sequences (at the cost of doubled storage)[15].

Here, to enable partition function calculations for even larger complexes, we define an overflow safe evaluation algebra that operates separately on the mantissa and exponent for the partition function calculation (Table 2.2b). The elements of the partition function recursion matrix are represented as $a = a_\mathrm{m} 2^{a_\mathrm{e}}$, where a_m is a single-precision (32-bit) floating point and a_e is a 32-bit integer, so the maximum expressible number is $\approx 10^{646457031}$.

For exposition, we assume in Table 2.2 that any expression is to be calculated with respect to a known reference exponent shift, γ, to which the expression is aligned. For instance, consider the expression $a \otimes b$ where $a = 40$ ($a_\mathrm{m} = 10$, $a_\mathrm{e} = 2$), $b = 96$ ($b_\mathrm{m} = 3$, $b_\mathrm{e} = 5$), and $\gamma = -6$, then $x_\mathrm{m} = a_\mathrm{m} \cdot b_\mathrm{m} = 10 \cdot 3 = 30$ and $x_\mathrm{e} = a_\mathrm{e} + b_\mathrm{e} + \gamma = 2 + 5 - 6 = 1$ corresponding to

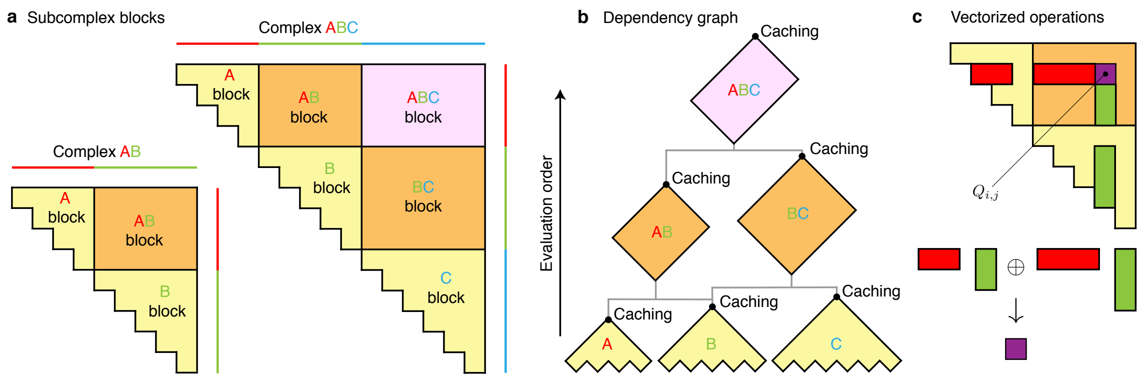


Figure 2.8: Blockwise operation order for dynamic programs operating on complex and test tube ensembles. (a) Subcomplex blocks within dynamic programming matrices (cf. Figure 2.5): triangular intrastrand and rectangular interstrand blocks. (b) Dependency graph for block evaluation: bottom to top for forward algorithms (depicted), top to bottom for backtracking algorithms. (c) Illustration of dot products from a representative step in a dynamic program ($Q_{i,j} \leftarrow \sum_{i \le d < j} Q_{i,d} Q_{d+1,j}$). Each combination of blocks gives a vectorizable dot product contribution; the case of $(d, d+1)$ spanning a strand break is skipped as it corresponds to a disconnected structure.

$a \otimes b = x_{\mathrm{m}} \cdot 2^{x_{\mathrm{e}}} \cdot 2^{-\gamma} = 30 \cdot 2^{1} \cdot 2^{6} = 3840$. See Section A.6.3 for a full description of the evaluation algebra including selection of an appropriate γ for each expression.

With this construction, the storage cost is thus identical to using double precision but overflow is no longer limiting, and the space and time complexity of the algorithm become the limiting factors. Empirically, we observe a ≈2–2.5× increase in cost for the overflow-safe evaluation algebra relative to a double-precision floating point evaluation algebra (Figure A.37). In practice, we use a blended approach by switching between the single-precision PFUNC, double-precision PFUNC, and single-precision OVERFLOWSAFE evaluation algebras as overflow occurs within a given complex (Section A.8.4).

2.3.7 Efficient Blockwise Dynamic Programs over Subcomplexes Using Caching and Vectorization

To this point, we have considered dynamic programs that operate on a complex of L strands. We now re-examine that goal in the more general context of a test tube ensemble containing the set of strand species Ψ^0 interacting to form the set of complex species Ψ. For example, suppose Ψ^0 contains M strand species and Ψ is defined to contain all complexes of up to $L_{\max}$ strands. The simplest option is to calculate the partition function for each complex $c \in \Psi$ independently[15]. With this approach, as described previously, the partition function $Q_{1,N}$ for a complex with N nucleotides is calculated with a dynamic program that builds up from short subsequences to the full-length sequence, sweeping along each diagonal of the matrix of subsequence partition functions (Figure 2.5). When multiple copies of the same strand species appear in a complex, intermediate results appear in multiple locations within the matrix. Moreover, when the same strand species appears in multiple complexes, intermediate results appear in multiple matrices.

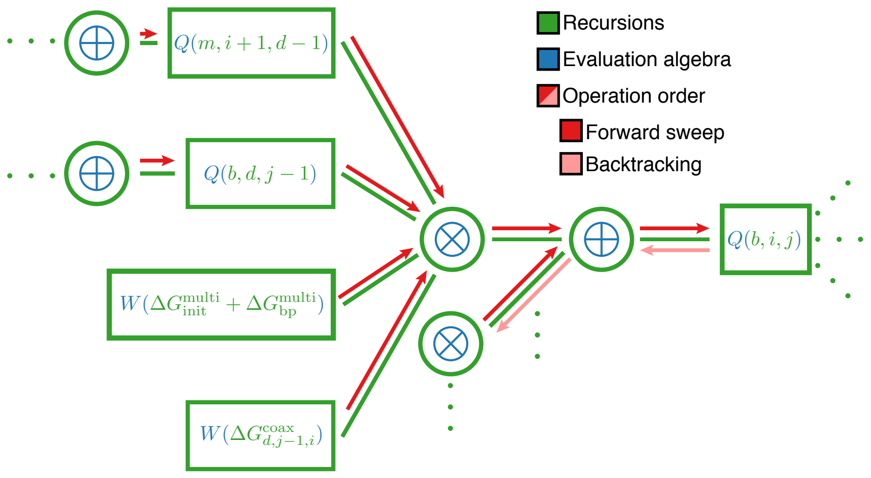

$Q(m, i+1, d-1)$
$Q(b, d, j-1)$
$W(\Delta G^{\text{multi}}_{\text{init}} + \Delta G^{\text{multi}}_{\text{bp}})$
$W(\Delta G^{\text{coax}}_{d,j-1,i})$
$Q(b, i, j)$
Recursions
Evaluation algebra
Operation order
Forward sweep
Backtracking

Figure 2.9: Conceptual interplay between three dynamic program ingredients: recursions, evaluation algebra, and operation order. Recursions specify the dependencies between subproblems and incorporate the details of the structural ensemble and free energy model. Evaluation algebras define the mathematical form of each subproblem. Operation orders specify the computational trajectory through the dependency graph of subproblems.

Here, we reduce the cost of calculating the partition functions for the set of complexes Ψ by decomposing each matrix into two types of subcomplex blocks (Figure 2.8a): triangular intrastrand blocks (e.g., blocks A, B, C) and rectangular interstrand blocks (e.g., blocks AB, BC, ABC). Blocks are computed in ascending order of the number of strands per block (blocks with the same number of strands can be calculated independently) and cached such that blocks arising in multiple locations within a complex or test tube ensemble are not recomputed. Section A.6.2 provides pseudocode for a blockwise operation order that is $O(N^3)$ for a complex of N nucleotides, including exact calculation of interior loop contributions[12,31]. Moreover, with this blockwise operation order, recursions (Figure 2.7) can be coded using vectorized dot products (Figure 2.8b) within triangular intrastrand blocks and rectangular interstrand blocks such that compilation with the appropriate evaluation algebra (Table 2.2) yields an efficient vectorized dynamic program for calculating the corresponding physical quantity. The interplay between the three dynamic programming ingredients (recursions, evaluation algebra, and operation order) is illustrated conceptually in Figure 2.9.

2.3.8 Enhanced Efficiency and Robustness of the Partition Function Algorithm for Complex Ensembles Including Very Large Complexes

Figure 2.10 highlights efficiency and robustness gains for partition function calculations on complex ensembles. Compared using the same physical model (without coaxial and dangle stacking), the vectorized NUPACK 4.0 implementation yields ≈30–90× speedups depending on the complex size. Furthermore, the overflow-safe evaluation algebra enables NUPACK 4.0 to perform partition function calculations on complexes containing 30,000 nt, while NUPACK 3.2 (using quadruple-precision) fails on the largest complex sizes due to overflow. Using the enhanced physical model that includes coaxial and dangle stacking subensembles, NUPACK 4.0 continues to achieve speedups of ≈13–45× over NUPACK 3.2 operating on the simpler physical model that neglects these terms.

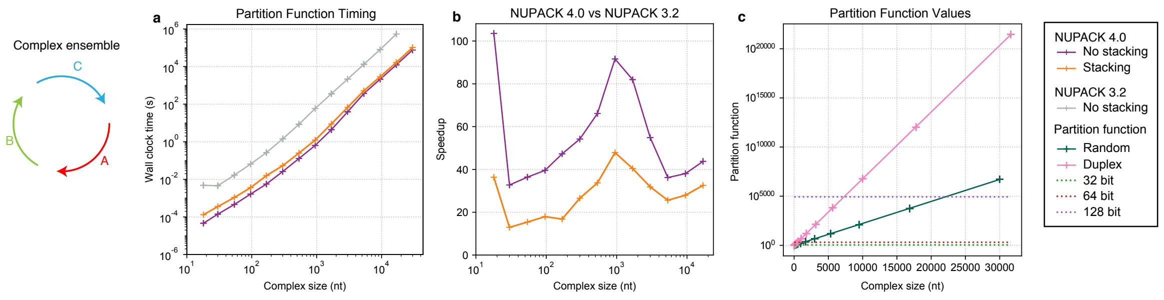


Figure 2.10: Enhanced efficiency and robustness for partition function calculations on complex ensembles including very large complexes. Calculation of the partition function for a complex of 3 strands, each with a different random sequence of uniform length. NUPACK 4.0 (vectorized, overflow-safe implementation, physical model with or without coaxial and dangle stacking) vs NUPACK 3.2 (not vectorized, quadruple-precision arithmetic, physical model with no coaxial or dangle stacking). (a) Computational cost. (b) Computational speedup (ratio of mean wall clock times). (c) Partition function values with coaxial and dangle stacking for random sequences and designed duplexes, with floating point overflow limits shown for reference (see Figure A.35). Means were taken over 5 sets of random sequences per ensemble size (results not available for largest complex size using NUPACK 3.0 due to overflow).

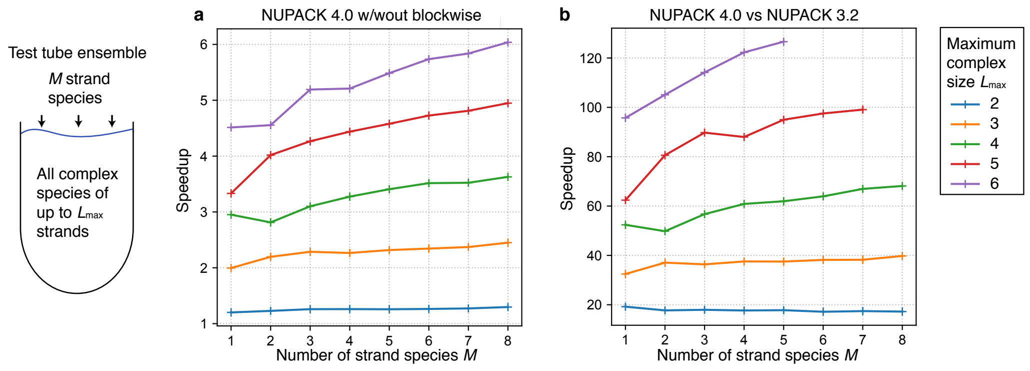


Figure 2.11: Enhanced efficiency of the partition function algorithm for sets of complexes in test tube ensembles. Calculation of the partition function for all complexes of up to $L_{\max}$ strands for a test tube ensemble containing $|\Psi^0|$ strand species, each with a different random 50 nt sequence. (a) Speedup with vs without blockwise caching for NUPACK 4.0 (Figure 2.8). (b) Speedup using NUPACK 4.0 (vectorized, blockwise caching, enhanced physical model with coaxial and dangle stacking) vs NUPACK 3.2 (no blockwise caching, not vectorized, physical model with no coaxial or dangle stacking). Mean wall clock times calculated over 10 sets of random sequences per ensemble size.

2.3.9 Enhanced Efficiency of the Partition Function Algorithm for Sets of Complexes in Test Tube Ensembles

Figure 2.11 highlights efficiency gains for partition function calculations for sets of complexes in test tube ensembles. Blockwise caching yields an empirical speedup of $\approx(L_{\max} - 1)$ for a range of test tube ensembles containing M strand species interacting to form all complexes of up to $L_{\max}$ strands (Figure 2.11a). Comparing the performance of NUPACK 4.0 (with the benefits of vectorization and block caching but the added cost of an enhanced physical model with coaxial and dangle stacking) to NUPACK 3.2 (without these features) reveals speedups of $\approx 20\times$ for test tubes containing all complexes of up to $L_{\max} = 2$ strands and $\approx 100\times$ for test tubes containing all complexes up to $L_{\max} = 6$ strands. With NUPACK 4.0, Figure 2.12 illustrates the size of test tube ensembles for which equilibrium analysis can be performed in ≤ 1 minute on a single computational core (e.g., $M = 80$

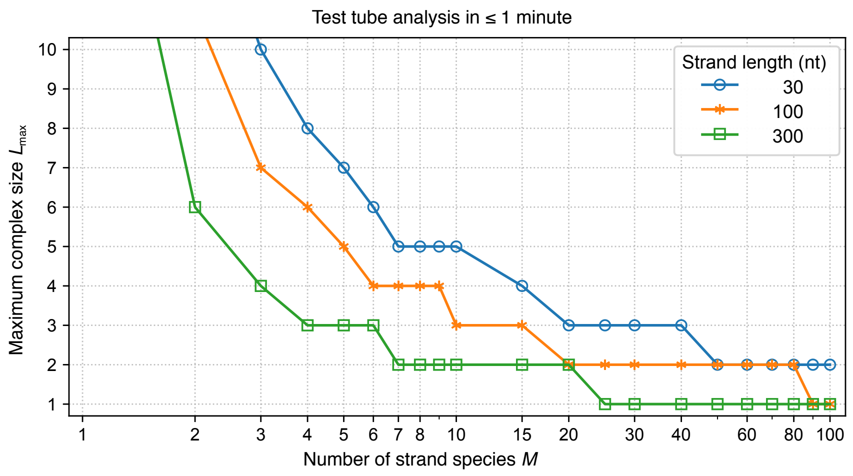


Figure 2.12: Equilibrium test tube analysis in under 1 minute. Calculation of the partition function and equilibrium complex concentration for a test tube ensemble containing M strand species that form all complexes of up to $L_{\max}$ strands. Symbols denote test tube ensembles for which the wall clock time $\leq$ 1 minute. After calculating the set of partition functions Q_Ψ for a given test tube ensemble Ψ, the set of equilibrium concentrations x_Ψ is obtained by solving the convex optimization problem (B.1). Mean wall clock time over 5 sets of random sequences per test tube ensemble size. Conditions: RNA, 37 °C, 1M Na$^+$, each strand introduced at 10 nM.

strand species of 100 nt each interacting to form all complex species of up to $L_{\max} = 2$ strands, or $M = 2$ strand species of 300 nt each interacting to form all complex species of up to $L_{\max} = 6$ strands).

2.3.10 Backtrack-Free Base-Pairing Probability Matrices

Historically, equilibrium base-pairing probabilities for a single strand[13,28] or a complex[15] are calculated using a dynamic program that backtracks through the matrix of subsequence partition functions. This backtracking process involves subtraction of intermediate partition function quantities, creating the risk of losing precision due to subtraction of large numbers differing by a small amount. To eliminate this concern, here we calculate equilibrium base-pairing probabilities without backtracking using the partition function evaluation algebra and a modification of the blockwise operation order.

To see how this is possible, consider a complex with strand ordering π = ABC and a total of N nucleotides. As an intermediate result, the partition function algorithm calculates $Q^b_{i,j}$, the conditional partition function for subsequence $i, \ldots, j$ subject to the constraint that i is paired to j. We may similarly calculate the conditional partition function, $Q^{b_{\text{ext}}}_{i,j}$, for the remaining nucleotides external to subsequence $i, \ldots, j$, namely nucleotides $j+1, \ldots, N, 1, \ldots, i-1$. Because the structural ensemble $\bar{\Gamma}$ excludes pseudoknots, the base pair $i \cdot j$ partitions the structural ensemble into non-interacting internal and external ensembles, so the partition function of all structures containing base pair $i \cdot j$ is the product $Q^b_{i,j} Q^{b_{\text{ext}}}_{i,j}$. As a result, the equilibrium probability of base pair $i \cdot j$ over ensemble $\bar{\Gamma}$ is given by

$$P_{i,j}(\phi) = Q^b_{i,j}(\phi) Q^{b_{\text{ext}}}_{i,j}(\phi) / Q_{1,N}(\phi). \tag{2.16}$$

Mathews employed this approach using new recursions to calculate the external conditional partition

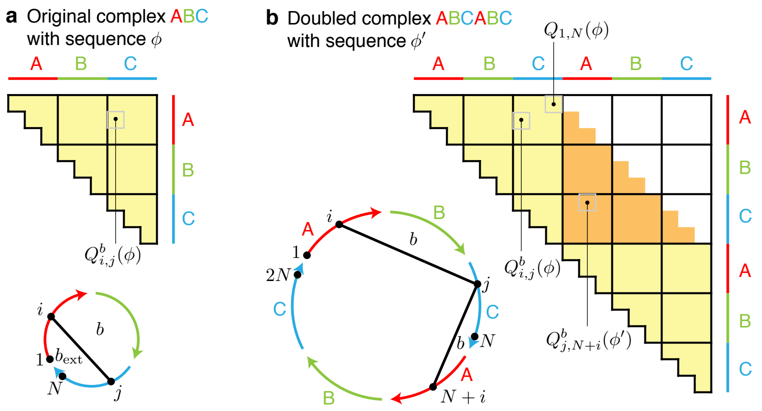


Figure 2.13: Backtrack-free calculation of the equilibrium base-pairing probability $P_{i,j}(\phi)$ for a complex ABC of N nucleotides with sequence ϕ using (2.17) and the conditional partition functions $Q^b_{i,j}(\phi)$ and $Q^b_{j,N+i}(\phi')$. The latter is calculated by considering the "doubled" complex ABCABC of $2N$ nucleotides with sequence ϕ'.

function $Q^{b_{\text{ext}}}_{i,j}$ for a single strand[8]. Here, treating the general case of a complex of L strands, we observe that $Q^{b_{\text{ext}}}_{i,j}$ can be calculated in a straightforward way without new recursions by replicating the strands to form a "doubled" complex with sequence ϕ' (e.g., ABCABC) containing $2N$ nucleotides and calculating $Q^b_{i,j}$ using the standard recursions for all subsequences of up to N nucleotides (Figure 2.13b). The external subsequence $j+1,\dots,N,1,\dots,i-1$ for the original complex with sequence ϕ is simply the internal subsequence $j, N+i$ for the doubled complex with sequence ϕ'. Hence, using the generic dynamic programming algorithm with the standard partition function evaluation algebra, we have:

$$P_{i,j}(\phi) = Q^b_{i,j}(\phi)Q^b_{j,N+i}(\phi')/Q_{1,N}(\phi). \tag{2.17}$$

In Figure 2.13, the yellow blocks are previously cached from the partition function calculation. The orange entries correspond to calculation of $Q^b_{j,N+i}(\phi')$. The cost of evaluating each entry is proportional to subsequence length (the horizontal or vertical distance from the diagonal), so the average cost per entry in the orange block is higher than for the yellow blocks. Empirically, after calculating the partition function Q at a cost C_Q, calculation of the pair probability matrix P costs an additional $C_P \approx$1.5–3C_Q (Figure A.36).

2.3.11 Evaluation Algebras and Backtracking Operation Orders for Simultaneous Structure Sampling, MFE Structure Determination, and Suboptimal Structure Determination

After calculating the partition function Q for a strand[29] or a complex[15], a structure s_{sample} can be randomly sampled from the structural ensemble $\overline{\Gamma}$ by backtracking through the matrix of subsequence partition functions. Likewise, after calculating the minimum free energy $\Delta G(\phi, s^{||}_{\text{MFE}})$ for a strand[30] or a complex[15], the corresponding MFE structure proxy $s_{\text{MFE}'}(\phi)$ can be determined by backtracking through the matrix of subsequence MFEs. These dynamic programs can be expressed in our unified dynamic programming framework (Figure 2.6) using the same set of recursion diagrams/equations (e.g., Figure 2.7) as the forward algorithms, but with the operation order reversed so the blockwise

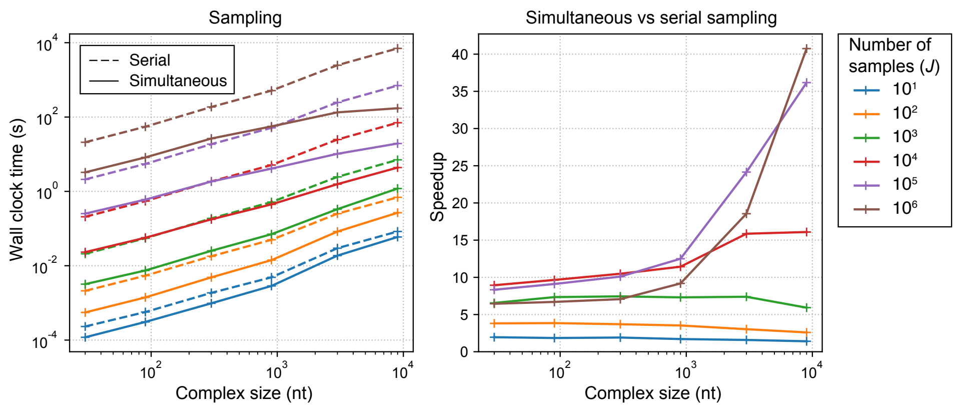


Figure 2.14: Enhanced efficiency for sampling multiple structures from complex ensembles using simultaneous rather than serial sampling. Comparison of runtimes for sequential sampling and simultaneous sampling. Sequences used range in length from 10 to 10^4 nucleotides, and between 10 and 10^6 samples were taken per sequence. Each data point represents an average over 10 sets of random sequences. See Section A.8.5 for additional data.

dependency tree (Figure 2.8b) is traversed top to bottom, and employing new evaluation algebras (Table 2.2cd).

For structure sampling, backtracking starts from the recursion for $Q_{1,N}$ and for MFE structure determination, backtracking starts from the recursion for $F_{1,N}$. In either case, backtracking is used to “choose” between competing recursion elements when a $\oplus$ operator is encountered and to “join” compatible recursions elements when a $\otimes$ operator is encountered; the mathematical implementations of these operators are described by quantity-specific evaluation algebras. For sampling, $\oplus$ corresponds to randomly choosing between competing (Boltzmann-weighted) recursion elements, while for MFE structure determination, $\oplus$ corresponds to choosing the MFE of competing recursion elements. For both structure sampling and MFE structure determination, $\otimes$ corresponds to the set union $\cup$ of compatible recursion elements.

The MFE structure determination algorithm can be generalized to calculate the set of suboptimal structures $\overline{\Gamma}(\phi, \Delta G_{\text{gap}})$ within a specified free energy gap $\Delta G_{\text{gap}} \geq 0$ of the MFE using generalized evaluation operators for $\oplus$ and $\otimes$ (see Section A.5.2). In practice, we implement this more general algorithm and then apply it with $\Delta G_{\text{gap}} = 0$ if the MFE structure is requested. The number of suboptimal structures can grow rapidly with ΔG_{gap} and N (Figure A.44) so we perform backtracking using a stack data structure that reduces memory usage by generating complete structures at the earliest opportunity, enabling these structures to be emitted in a streaming fashion while additional structures are determined (see Section A.6.6).

While the pair probability matrix P provides the equilibrium probability of each base pair over the complex ensemble, it does not reveal correlation information between different base pairs. By sampling a set of J secondary structures and averaging or clustering over this set, it is possible to address questions like “what is the probability that a set of adjacent bases are simultaneously unpaired?”[29] or “is the free energy landscape dominated by multiple deep basins each defined by a set of related secondary structures?”[32]. Existing algorithms perform serial sampling of J structures for a strand[29] ($O(JN^2)$ excluding long interior loops) or a complex[15] ($O(JN^3)$ with exact treatment of interior loops). Motivated by the central use case where a set of J structures is needed for averaging or clustering, here we develop a simultaneous sampling approach that samples J structures all at once ($O(JN^2)$ with exact treatment of interior loops). A given recursion element may contribute to a large number of sampled structures (a circumstance that is augmented for designed free energy landscapes containing deep wells), so we perform backtracking using a priority queue data structure that reduces computational effort by ensuring that all samples of any given recursion element are performed during a single visit to that recursion element (Section A.6.5). With the simultaneous sampling algorithm, we observe order of magnitude speedups over serial sampling as J increases above $\approx 10^3$ (Figure 2.14), and empirical complexity $\sim J^{0.8}N^1$ for J samples from a complex ensemble with N nucleotides (Section A.8.5).

2.4 Conclusions

The new unified dynamic programming framework combines recursions capturing the details of the physical model with quantity-specific evaluation algebras and operation orders to enable efficient and robust calculation of diverse physical quantities over complex and test tube ensembles of interacting DNA or RNA strands. The physical model was upgraded by deriving recursions for the complex ensemble that include coaxial and dangle stacking subensembles for multiloops and exterior loops. The recursions are coded generically and then compiled with a quantity-specific evaluation algebra and operation order to generate an executable for each physical quantity. As a result, future upgrades to the physical model can be implemented by updating the generic recursions rather than by updating code for each physical quantity. For large complexes, robustness is achieved for partition function calculations using an overflow-safe evaluation algebra, and for equilibrium pair probabilities by using a backtrack-free operation order, enabling calculations on complexes containing 30,000 nt. For test tube ensembles, dramatic efficiency gains of 1–2 orders of magnitude are achieved using a new blockwise operation order that facilitates vectorization and caching. Recognizing that Boltzmann sampling is most useful for averaging or clustering information calculated on large set of structures, a new sampling algorithm yields order-of-magnitude speedups by sampling all requested structures simultaneously. These enhancements to the physical model, algorithm robustness, and algorithm speed, are directly applicable to sequence design over complex and test tube ensembles[16–18] as sequence analysis is the foremost computational cost of sequence design; work is underway to integrate these advances into the NUPACK 4.0 sequence design algorithms.

2.5 Methods Summary

2.5.1 Implementation.

NUPACK algorithms are programmed in the C++17 programming language. Dynamic programs are implemented using a generic programming paradigm[33] employing expression templates and compile-time polymorphism; generic recursion equations capturing the details of the structural ensemble and free energy model are translated via template metaprogramming into a separate vectorized executable for calculating each physical quantity in Table 2.2. Single-threaded single instruction multiple data (SIMD) vectorization is implemented using the Boost.SIMD library[34]. The convex optimization problem (B.1) is solved in the dual form using an efficient trust region method[15] using the Armadillo linear algebra library for matrix operations[35].

2.5.2 Trials

All benchmarks were run on AWS EC2 C5 instances (3.0 GHz Intel Xeon Platinum processors) with 72 GB of memory (except 144 GB for Figure 2.10).

2.6 Resources

2.6.1 NUPACK Source Code

The NUPACK source code can be downloaded for non-commercial academic use subject to the NUPACK License (`nupack.org`). NUPACK documentation includes a detailed User Guide and a directory containing example analysis and design jobs.

2.6.2 NUPACK Python Module

The all-new NUPACK Python interface allows streamlined and flexible in-memory scripting of NUPACK jobs, reducing file I/O and increasing the convenience of developing workflows composing multiple NUPACK commands. For backward compatibility, Python scripts are provided that can be integrated into existing workflows in place of previous releases of NUPACK executables, taking advantage of the dramatically improved performance of NUPACK 4.0 implementations.

2.6.3 Support

Please direct questions, comments, feature requests, and bug reports to `support@nupack.org`.

2.7 Author Information

Corresponding Author

E-mail: niles@caltech.edu

Author Contributions

*M.E.F and N.J.P. contributed equally.

Notes

The authors declare no competing financial interests.

2.8 Acknowledgments

We thank all the NUPACK users that have helped out as beta testers over the years, as well as the many NUPACK users that have emailed `support@nupack.org` to request features or report bugs. We thank J.S. Bois for helpful discussions. This work was funded by the National Science Foundation (Software Elements NSF-OAC-1835414, INSPIRE NSF-CHE-1643606, Molecular Programming

Project NSF-CCF-1317694), by the Programmable Molecular Technology Center (PMTC) within the Beckman Institute at Caltech, by the AWS/IST Cloud Credit Program at Caltech, by a Microsoft Azure sponsorship, by the National Institutes of Health (National Research Service Award T32 GM007616), by a Professorial Fellowship at Balliol College, University of Oxford, and by the Eastman Visiting Professorship at the University of Oxford.

Bibliography

[1] J. N. Zadeh, C. D. Steenberg, J. S. Bois, B. R. Wolfe, M. B. Pierce, A. R. Khan, R. M. Dirks, and N. A. Pierce. “NUPACK: Analysis and Design of Nucleic Acid Systems”. In: *J. Comput. Chem.* 32.1 (2011), pp. 170–173. DOI: 10.1002/jcc.21596.

[2] I. Tinoco Jr., O. Uhlenbeck, and M. Levine. “Estimation of Secondary Structure in Ribonucleic Acids”. In: *Nature* 230 (1971), pp. 362–367.

[3] M. J. Serra and D. H. Turner. “Predicting Thermodynamic Properties of RNA”. In: *Methods Enzymol.* 259 (1995), pp. 242–261.

[4] J. SantaLucia Jr. “A Unified View of Polymer, Dumbbell, and Oligonucleotide DNA Nearest-Neighbor Thermodynamics”. In: *Proc. Natl. Acad. Sci. U. S. A.* 95.4 (1998), pp. 1460–1465.

[5] D. H. Mathews, J. Sabina, M. Zuker, and D. H. Turner. “Expanded Sequence Dependence of Thermodynamic Parameters Improves Prediction of RNA Secondary Structure”. In: *J. Mol. Biol.* 288 (1999), pp. 911–940.

[6] M. Zuker. “Mfold Web Server for Nucleic Acid Folding and Hybridization Prediction”. In: *Nucleic Acids Res.* 31.13 (2003), pp. 3406–3415.

[7] J. SantaLucia Jr. and D. Hicks. “The Thermodynamics of DNA Structural Motifs”. In: *Annu. Rev. Biophys. Biomol. Struct.* 33 (2004), pp. 415–440. ISSN: 1056-8700.

[8] D. H. Mathews, M. D. Disney, J. L. Childs, S. J. Schroeder, M. Zuker, and D. H. Turner. “Incorporating Chemical Modification Constraints into a Dynamic Programming Algorithm for Prediction of RNA Secondary Structure”. In: *Proc. Natl. Acad. Sci. U. S. A.* 101.19 (2004), pp. 7287–7292. ISSN: 0027-8424. DOI: 10.1073/pnas.0401799101.

[9] R. T. Koehler and N. Peyret. “Thermodynamic Properties of DNA Sequences: Characteristic Values for the Human Genome”. In: *Bioinformatics* 21.16 (2005), pp. 3333–3339. ISSN: 1367-4803. DOI: Doi10.1093/Bioinformatics/Bti530.

[10] Z. J. Lu, D. H. Turner, and D. H. Mathews. “A Set of Nearest Neighbor Parameters for Predicting the Enthalpy Change of RNA Secondary Structure Formation”. In: *Nucleic Acids Res.* 34.17 (2006), pp. 4912–4924. ISSN: 0305-1048. DOI: 10.1093/nar/gkl472.

[11] R. Tyagi and D. H. Mathews. “Predicting Helical Coaxial Stacking in RNA Multibranch Loops”. In: *RNA* 13.7 (2007), pp. 939–951. ISSN: 1355-8382. DOI: 10.1261/rna.305307.

[12] R. M. Dirks and N. A. Pierce. “A Partition Function Algorithm for Nucleic Acid Secondary Structure Including Pseudoknots”. In: *J. Comput. Chem.* 24 (2003), pp. 1664–1677.

[13] R. M. Dirks, M. Lin, E. Winfree, and N. A. Pierce. “Paradigms for Computational Nucleic Acid Design”. In: *Nucleic Acids Res.* 32.4 (2004), pp. 1392–1403.

[14] R. M. Dirks and N. A. Pierce. “An Algorithm for Computing Nucleic Acid Base-Pairing Probabilities Including Pseudoknots”. In: *J. Comput. Chem.* 25 (2004), pp. 1295–1304.

[15] R. M. Dirks, J. S. Bois, J. M. Schaeffer, E. Winfree, and N. A. Pierce. “Thermodynamic Analysis of Interacting Nucleic Acid Strands”. In: *SIAM Rev.* 49.1 (2007), pp. 65–88.

[16] J. N. Zadeh, B. R. Wolfe, and N. A. Pierce. “Nucleic Acid Sequence Design via Efficient Ensemble Defect Optimization”. In: *J. Comput. Chem.* 32 (2011), pp. 439–452. DOI: 10.1002/jcc.21633.

[17] B. R. Wolfe and N. A. Pierce. “Nucleic Acid Sequence Design for a Test Tube of Interacting Nucleic Acid Strands”. In: *ACS Synth. Biol.* 4.10 (2015), pp. 1086–1100.

[18] B. R. Wolfe, N. J. Porubsky, J. N. Zadeh, R. M. Dirks, and N. A. Pierce. “Constrained Multistate Sequence Design for Nucleic Acid Reaction Pathway Engineering”. In: *J. Am. Chem. Soc.* 139 (2017), pp. 3134–3144.

[19] N. J. Porubsky, M. E. Fornace, and N. A. Pierce. “NUPACK 4.0: Analysis and Design of Nucleic Acid Structures, Devices, and Systems”. In: *Submitted* ().

[20] L. M. Hochrein, M. Schwarzkopf, M. Shahgholi, P. Yin, and N. A. Pierce. “Conditional Dicer Substrate Formation via Shape and Sequence Transduction with Small Conditional RNAs”. In: *J. Am. Chem. Soc.* 135.46 (2013), pp. 17322–17330.

[21] C. Geary, P. W. K. Rothemund, and E. S. Andersen. “A Single-Stranded Architecture for Cotranscriptional Folding of RNA Nanostructures”. In: *Science* 345.6198 (2014), pp. 799–804.

[22] T. Xia, J. SantaLucia Jr., M. Burkard, R. Kierzek, S. Schroeder, X. Jiao, C. Cox, and D. Turner. “Thermodynamic Parameters for an Expanded Nearest-Neighbor Model for Formation of RNA Duplexes with Watson-Crick Base Pairs”. In: *Biochemistry* 37.42 (1998), pp. 14719–14735.

[23] D. H. Turner and D. H. Mathews. “NNDB: The Nearest Neighbor Parameter Database for Predicting Stability of Nucleic Acid Secondary Structure”. In: *Nucleic Acids Res.* 38 (2010), pp. D280–D282. ISSN: 0305-1048. DOI: 10.1093/nar/gkp892.

[24] V. Bloomfield, D. Crothers, and I. Tinoco Jr. *Nucleic Acids: Structures, Properties, and Functions*. Sausalito, CA: University Science Books, 2000.

[25] D. H. Turner and D. H. Mathews. “NNDB: the nearest neighbor parameter database for predicting stability of nucleic acid secondary structure”. In: *Nucleic Acids Res.* 38 (2010), pp. D280–D282.

[26] N. Peyret. “Prediction of Nucleic Acid Hybridization: Parameters and Algorithms”. Thesis. 2000.

[27] S. Bommarito, N. Peyret, and J. SantaLucia. “Thermodynamic Parameters for DNA Sequences with Dangling Ends”. In: *Nucleic Acids Res.* 28.9 (2000), pp. 1929–1934. ISSN: 0305-1048. DOI: DOI10.1093/nar/28.9.1929.

[28] J. McCaskill. “The Equilibrium Partition Function and Base Pair Binding Probabilities for RNA Secondary Structure”. In: *Biopolymers* 29 (1990), pp. 1105–1119.

[29] Y. Ding and C. Lawrence. “A Statistical Sampling Algorithm for RNA Secondary Structure Prediction”. In: *Nucleic Acids Res.* 31.24 (2003), pp. 7280–7301.

[30] M. Zuker and P. Stiegler. “Optimal Computer Folding of Large RNA Sequences Using Thermodynamics and Auxiliary Information”. In: *Nucleic Acids Res.* 9.1 (1981), pp. 133–147.

[31] R. B. Lyngsø, M. Zuker, and C. N. S. Pedersen. “Fast Evaluation of Internal Loops in RNA Secondary Structure Prediction”. In: *Bioinformatics* 15.6 (1999), pp. 440–445.

[32] Y. Ding, C. Chan, and C. Lawrence. “RNA Secondary Structure Prediction by Centroids in a Boltzmann Weighted Ensemble”. In: *RNA* 11 (2005), pp. 1157–1166.

[33] A. A. Stepanov and D. E. Rose. *From Mathematics to Generic Programming*. Crawfordsville, Indiana: Pearson Education, 2014.

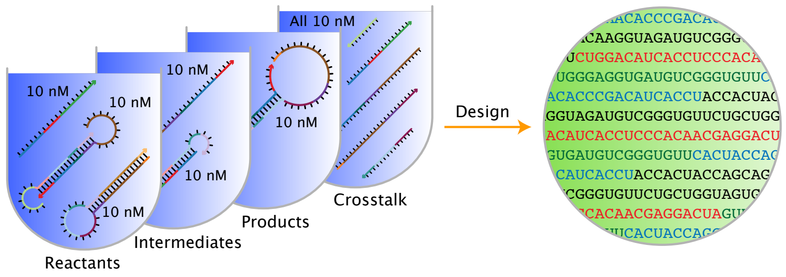
[34] P. Esterie, J. Falcou, M. Gaunard, and J.-T. Lapreste. “Boost.SIMD: Generic Programming for Portable SIMDization”. In: *Proceedings of the 2014 Workshop on Programming Modles for SIMD/Vector Processing*. Bh0krTimes Cited:3Cited References Count:4International Conference on Parallel Architectures and Compilation Techniques. New York: ACM, 2012, pp. 1–8. ISBN: 1089-795x.

[35] C. Sanderson and R. Curtin. “Armadillo: A Template-Based C++ Library for Linear Algebra”. In: *J. Open Source Softw.* 1 (2016), p. 26.

Chapter 3

Constrained Multistate Sequence Design for Nucleic Acid Reaction Pathway Engineering

This chapter was adapted from material in B. R. Wolfe*, N. J. Porubsky*, J. N. Zadeh, R. M. Dirks, and N. A. Pierce. "Constrained Multistate Sequence Design for Nucleic Acid Reaction Pathway Engineering". In: *Journal of the American Chemical Society* 139.8 (2017), pp. 3134–3144. ISSN: 15205126. DOI: 10.1021/jacs.6b12693. and the supplementary information thereof.



We describe a framework for designing the sequences of multiple nucleic acid strands intended to hybridize in solution via a prescribed reaction pathway. Sequence design is formulated as a multistate optimization problem using a set of target test tubes to represent reactant, intermediate, and product states of the system, as well as to model crosstalk between components. Each target test tube contains a set of desired "on-target" complexes, each with a target secondary structure and target concentration, and a set of undesired "off-target" complexes, each with vanishing target concentration. Optimization of the equilibrium ensemble properties of the target test tubes implements both a positive design paradigm, explicitly designing for on-pathway elementary steps, and a negative design paradigm, explicitly designing against off-pathway crosstalk. Sequence design is performed subject to diverse user-specified sequence constraints including composition constraints, complementarity constraints, pattern constraints, and biological constraints. Constrained multistate sequence design facilitates nucleic acid reaction pathway engineering for diverse applications in molecular programming and synthetic biology. Design jobs can be run online via the NUPACK web application.

3.1 Introduction

Life is orchestrated by programmable biopolymers – DNA, RNA, and proteins – that execute complex self-assembly and disassembly processes to grow, regulate, and repair organisms. The emerging discipline of molecular programming is inspired by these biological proofs of principle and seeks to establish sequence design principles and algorithms that enable robust encoding of a desired molecular function into biopolymer sequences. To engineer dynamic self-assembly and disassembly processes, it is necessary to control not just equilibrium properties, but the kinetic pathways by which molecules interact. During the last decade, the programmable chemistry of nucleic acid base pairing has provided a fertile design space for engineering pathway-controlled self-assembly and disassembly processes[1,2].

Molecular programmers engineer nucleic acid reaction pathways using an ever-increasing variety of small conditional DNA and RNA motifs (scDNAs and scRNAs) that exploit diverse design elements to interact and change conformation via prescribed hybridization cascades[1,2]. Modes of nucleating interactions include toehold/toehold[3–10], loop/toehold[11,12], loop/loop[13,14], and template/toehold[12] hybridization. Modes of strand displacement include 3-way branch migration[3–5,7–11], 4-way branch migration[6,12,14,15], and spontaneous dissociation[7,10,12]. To exert control over the order of self-assembly and disassembly events, scDNAs are designed to co-exist metastably (i.e., the molecules are kinetically trapped) or stably (i.e., the molecules are thermodynamically trapped), with the next step in the reaction pathway triggered either by a cognate molecular input detected from the environment or by a molecular output of a previous step in the reaction pathway. Principles for engineering conditional metastability include nucleation barriers[4,8], topological constraints[13,14], toehold sequestration[5,7,8,10,12], and template unavailability[12], while principles for engineering conditional stability include cooperativity[16] and sequence transduction[12]. These pathway-controlled self-assembly and disassembly reactions have been driven by the enthalpy of base pairing[3–6,8,10–13] and the entropy of mixing super[7,8,10,12]. These design elements have enabled the rational design and construction of scDNAs executing diverse dynamic functions, including catalysis, signal amplification, sequence transduction, shape transduction, Boolean logic, and locomotion[1,2].

Devising a new reaction pathway is akin to molecular choreography, requiring conception of both the scRNA participants (or equivalently scDNA participants) and the dance that they will execute via pathway-controlled self-assembly and disassembly operations. Owing to the modularity of scRNA function, new reaction pathways representing complex dynamic functions can often be fruitfully sketched by hand. Once a new reaction pathway has been choreographed, the task remains of encoding the intended pathway-controlled interactions and conformational changes into the sequences of the constituent scRNAs. To program this dynamic function, the nucleic acid sequences must be designed so that the molecules predominantly execute the desired on-pathway interactions while avoiding off-pathway alternatives. Here, we address the dual challenges of formulating and solving the sequence design problem for nucleic acid reaction pathway engineering.

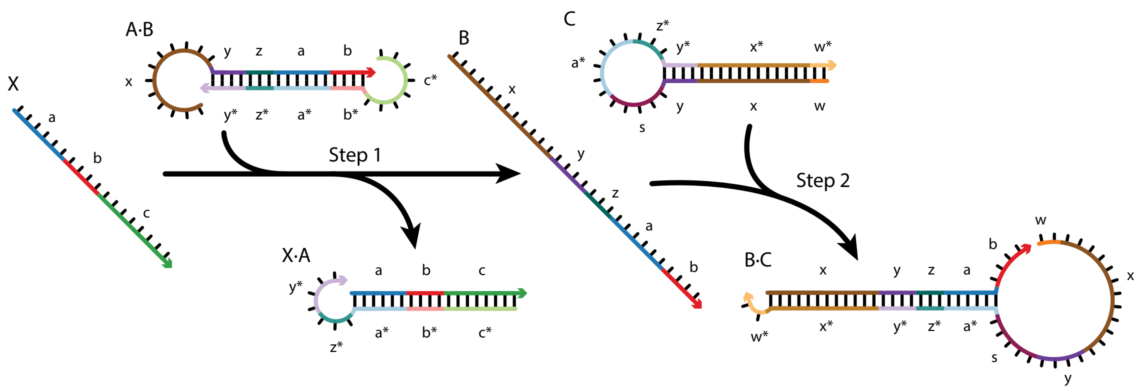


Figure 3.1: Reaction pathway for conditional Dicer substrate formation via shape and sequence transduction with small conditional RNAs (scRNAs)[12]. scRNA A·B detects input X (comprising sequence 'a-b-c'), leading to production of Dicer substrate B·C (targeting independent sequence 'w-x-y-z'). Step 1: X displaces A from B via toehold-mediated 3-way branch migration and spontaneous dissociation. Step 2: B assembles with C via loop/toehold nucleation and 3-way branch migration to form Dicer substrate B·C. See reaction pathways for additional case studies in Section B.2.1.

3.2 Design Formulation

3.2.1 Reaction Pathway Specification

Consider a set of nucleic acid molecules intended to execute a prescribed hybridization cascade. For example, the reaction pathway of Figure 3.1 describes scRNAs that upon binding to input X, perform shape and sequence transduction to form a Dicer substrate targeting an independent output Y for silencing[12]. A reaction pathway specifies the *elementary steps* (each a self-assembly or disassembly operation in which complexes form or break) by which the molecules are intended to interact, the desired secondary structure for each on-pathway complex, and the complementarity relationships between sequence domains in the molecules. For example, in the reaction pathway of Figure 3.1, there are two elementary steps (Step 1: X + A·B → X·A + B, Step 2: B + C → B·C) involving six on-pathway complexes (X, A·B, X·A, B, C, B·C) and numerous sequence domains ('a*' complementary to 'a', 'b*' complementary to 'b', and so on).

In addition to specifying a set of desired on-pathway elementary steps, each reaction pathway also implicitly specifies a much larger set of off-pathway interactions, corresponding to undesired *crosstalk* between components within the pathway or with components from other unrelated reaction pathways. To perform sequence design for reaction pathway engineering, we formulate a multistate optimization problem to explicitly design for on-pathway elementary steps (a positive design paradigm) and against off-pathway crosstalk (a negative design paradigm).

3.2.2 Multistate Test Tube Design Ensemble

A multistate test tube design problem is specified as a set of target test tubes, Ω. Each tube, $h \in \Omega$, contains a set of desired *on-target complexes*, Ψ_h^{on}, and a set of undesired *off-target complexes*, Ψ_h^{off}. For each on-target complex, $j \in \Psi_h^{\text{on}}$, the user specifies a target secondary structure, s_j, and a target concentration, $y_{h,j}$. For each off-target complex, $j \in \Psi_h^{\text{off}}$, the target concentration is vanishing ($y_{h,j} = 0$) and there is no target structure ($s_j = \emptyset$). The set of complexes in tube h is then

$\Psi_h \equiv \Psi_h^{\text{on}} \cup \Psi_h^{\text{off}}$ and the set of all complexes in multistate test tube ensemble Ω is $\Psi \equiv \cup_{h\in\Omega}\Psi_h$.

Consider specification of the multistate test tube ensemble, Ω, for the design of N orthogonal systems for a reaction pathway of M elementary steps. One *elementary step tube* is specified for each step $m = 0, \ldots, M$ for each system $n = 1, \ldots, N$ (treating formation of the initial reactants as a precursor "Step 0"). Additionally, a single *global crosstalk tube* is specified to minimize off-pathway interactions between the reactive species generated during all elementary steps of all systems. The total number of target test tubes is then $|\Omega| = N * (M + 1) + 1$.

A detailed description of our approach for specifying target test tubes is provided in Section 3.2.3.

3.2.3 Specification of Target Test Tubes

General Formulation

Consider specification of the multistate test tube ensemble, Ω, for the design of N orthogonal systems for a reaction pathay of M elementary steps, each corresponding to a self-assembly or disassembly operation in which complexes form or break. One elementary step tube is specified for each step $m = 0, 1, \ldots, M$ for each system $n = 1, \ldots, N$ (treating formation of the initial reactants as a precursor "Step 0"). Additionally, a single global crosstalk tube is specified to minimize off-pathway interactions between the reactive species generated during all elementary steps of all systems. The total number of target test tubes is then $|\Omega| = (M + 1) \times N + 1$.

Elementary Step Tubes

Consider elementary step m for orthogonal system n with on-pathway products $\Psi_{m_n}^{\text{products}}$ that are intended to form at non-zero concentrations at equilibrium, and reactants $\Psi_{m_n}^{\text{reactants}}$ that are intended to fully convert into the on-pathway products at equilibrium. Furthermore, consider the set of off-pathway products, $\Psi_{m_n}^{\text{crosstalk}}$, corresponding to unintended interactions between these same reactants.

The *elementary step tube* for step m of system n is then:

$$\boxed{\text{Step } m_n \text{ tube: } \Psi_h^{\text{on}} \equiv \Psi_{m_n}^{\text{products}}, \quad \Psi_h^{\text{off}} \equiv \Psi_{m_n}^{\text{reactants}} \cup \Psi_{m_n}^{\text{crosstalk}}}$$

where the on-targets are the on-pathway products, and the off-targets are the reactants and off-pathway crosstalk products. For step m of system n, this tube designs for full conversion of cognate reactants into cognate products and against local crosstalk between these same reactants. One elementary step tube is specified for each elementary step $m = 0, 1, \ldots, M$ for each system $n = 1, \ldots, N$.

The off-pathway crosstalk products for step m of system n are defined as:

$$\boxed{\Psi_{m_n}^{\text{crosstalk}} = \Psi_{m_n}^{L\leq L_{\max}} - \Psi_{m_n}^{\text{exclude}}}$$

where the set $\Psi_{m_n}^{L\leq L_{\max}}$ denotes the set of all complexes of up to $L_{\max}$ strands (that are not already on-targets in the Step m_n tube). The set $\Psi_{m_n}^{\text{exclude}}$ contains energetically favorable complexes that we wish to exclude from the ensemble for the current elementary step (e.g., downstream on-pathway products, or off-pathway products that are inhibited kinetically rather than thermodynamically, and hence are not suitable for inclusion in the equilibrium optimization ensemble).

Global Crosstalk Tube

To actively design against global crosstalk, we additionally specify a single *global crosstalk tub*:

$$\boxed{\text{Global crosstalk tube: } \Psi_h^{\text{on}} \equiv \Psi_{\text{global}}^{\text{reactive}}, \quad \Psi_h^{\text{off}} \equiv \Psi_{\text{global}}^{\text{crosstalk}}}$$

where $\Psi_{\text{global}}^{\text{reactive}}$ denotes the set of all reactive species generated during all elementary steps for all systems and $\Psi_{\text{global}}^{\text{crosstalk}}$ denotes the set of undesired crosstalk products resulting from interactions between these species.

For the global crosstalk tube, we exploit *motif simplification* to enable specification of the on-target and off-target complexes using using only monomers and dimers. The presumption is that motif complexity will typically decrease rather than increase crosstalk between reactive species, so that for the global crosstalk tube, motif simplification is justified in the service of efficiency and simplicity. By contrast, for the elementary step tubes, reactant and product complexes are treated without motif simplification, ensuring that any energetic effects associated with the full complexes (either unfavorable [e.g., 3-arm junction for CHA product] or favorable [e.g., nick stack for HCR]) are taken into consideration.

To define various forms of motif simplification, it is helpful to define input and output domains that participate in the elementary steps. Each scRNA or scDNA motif (monomer, dimer, trimer, etc) has one or more *input domains* that control the state of one or more *output domains*. An inactive output domain is toggled to the active state when sequestering input domains hybridize to active output domains generated by earlier elementary steps in the reaction pathway. Nucleation with an input domain occurs via hybridization to an accessible loop or toehold. Targets that serve as inputs to a reaction pathway may be viewed as unconditionally active output domains that are available to hybridize to complementary input domains at any step in a reaction pathway.

Using motif simplification, we specify the *reactive species* and *cognate products* for system n as follows:

- $\lambda_n^{\text{simple}}$: scRNA and scDNA motifs with multiple input or output domains are simplified so that only the input and output domains for a single elementary step are present in each simplified motif.
- $\lambda_n^{\text{ss-out}}$: single-stranded output domains are specified for each elementary step, removing other concatenated or hybridized domains that represent the history or future of the reaction (participating in previous or future elementary steps).

- $\lambda_n^{\text{ss-in}}$: single-stranded nucleation sites within input domains (toeholds or loops) are specified isolated from the surrounding domains representing the history or future of the reaction.*

- $\lambda_n^{\text{reactive}} \equiv \{\lambda^{\text{simple}} \cup_{\text{simp}} \lambda^{\text{ss-out}} \cup_{\text{simp}} \lambda^{\text{ss-in}}\}_n$: the set of reactive species for system n is specified using a union operator $\cup_{\text{simp}}$ that eliminates redundancies when one monomer species is an accessible subsequence of another monomer species.

- $\lambda_n^{\text{cognate}}$: cognate products expected to form from reactive species in $\lambda_n^{\text{reactive}}$ based on sequence complementarity imposed by the reaction pathway (e.g., an input domain within a motif in $\lambda_n^{\text{simple}}$ is expected to hybridize to a complementary output domain in $\lambda_n^{\text{ss-out}}$).

These definitions for $\lambda_n^{\text{reactive}}$ and $\lambda_n^{\text{cognate}}$ are then used to define the on-targets for the global crosstalk tube:

$$\boxed{\Psi_{\text{global}}^{\text{reactive}} \equiv \cup_{n=1,\ldots,N}\{\lambda_n^{\text{reactive}}\}}$$

and the off-targets for the global crosstalk tube:

$$\boxed{\Psi_{\text{global}}^{\text{crosstalk}} \equiv \Psi_{\text{global}}^{L \le L_{\max}} - \cup_{n=1,\ldots,N}\{\lambda_n^{\text{cognate}}\}}$$

Here, $\Psi_{\text{global}}^{L \le L_{\max}}$ denotes the set of all complexes of up to $L_{\max}$ strands (that are not already on-targets in the global crosstalk tube). The set $\cup_{n=1,\ldots,N}\{\lambda_n^{\text{cognate}}\}$ contains all the cognate products that the reactive species in the N orthogonal systems are expected to form based on sequence complementarity. Crucially, by excluding these cognate products from $\Psi_{\text{global}}^{\text{crosstalk}}$, they do not appear in the global crosstalk tube as either on-targets or off-targets. Hence, all reactive species in the global crosstalk tube are forced to perform either no reaction (remaining as desired on-targets) or to undergo a crosstalk reaction (forming undesired off-targets), providing the basis for minimization of global crosstalk during sequence optimization.

Target Test Tube Specification for Conditional Dicer Substrate Formation via Shape and Sequence Transduction with scRNAs

Target test tubes are defined using the specification of Section 3.2.3 with the following definitions. The total number of target test tubes is $|\Omega| = \sum_{n=1,\ldots,N}$ {Step 0, Step 1, Step 2}$_n$ + Crosstalk = $3N + 1$; the target test tubes in the multistate test tube ensemble, Ω, are indexed by $h = 1, \ldots, 3N + 1$. $L_{\max} = 2$ for all tubes.

*The role of $\lambda_n^{\text{ss-in}}$ is to enable $\Psi_{\text{global}}^{\text{crosstalk}}$ (the off-targets for the global crosstalk tube) to be specified without requiring any complexes larger than dimers. For example, consider a dimer motif with an exposed toehold. Crosstalk via kissing of this toehold with that of another dimer motif would yield a tetramer off-target; including these toeholds as isolated monomers in $\lambda_n^{\text{ss-in}}$ allows this crosstalk interaction to be described by an off-target dimer, which is automatically included in $\Psi_{\text{global}}^{\text{crosstalk}}$ by considering all off-targets of up to $L_{\max} = 2$ strands. Further, inclusion of loop nucleation sites in $\lambda_n^{\text{ss-in}}$ enables designing against pseudoknotted toehold/loop and loop/loop crosstalk interactions without needing to explicitly include pseudoknots in the structural ensemble of any complex.

Reactants for System *n*

- Target: X_n
- scRNAs: $\{A{\cdot}B, C\}_n$

Elementary Step Tubes for System *n*

- Step 0_n: $\Psi_{0_n}^{\text{products}} \equiv \{X, A{\cdot}B, C\}_n$; $\Psi_{0_n}^{\text{reactants}} \equiv \{A, B{\cdot}C\}_n$; $\Psi_{0_n}^{\text{exclude}} \equiv \{X{\cdot}A\}$
- Step 1_n: $\Psi_{1_n}^{\text{products}} \equiv \{X{\cdot}A, B\}_n$; $\Psi_{1_n}^{\text{reactants}} \equiv \{X, A{\cdot}B\}_n$; $\Psi_{1_n}^{\text{exclude}} \equiv \emptyset$
- Step 2_n: $\Psi_{2_n}^{\text{products}} \equiv \{B{\cdot}C\}_n$; $\Psi_{2_n}^{\text{reactants}} \equiv \{B, C\}_n$; $\Psi_{2_n}^{\text{exclude}} \equiv \emptyset$

Crosstalk Tube

- Crosstalk tube: $\Psi_{\text{global}}^{\text{reactive}} \equiv \cup_{n=1,\ldots,N}\{\lambda_n^{\text{reactive}}\}$; $\Psi_{\text{global}}^{\text{crosstalk}} \equiv \Psi_{\text{global}}^{L \le L_{\text{max}}} - \cup_{n=1,\ldots,N}\{\lambda_n^{\text{cognate}}\}$

The reactive species and cognate products for system *n* are:

- $\lambda_n^{\text{simple}} = \{A{\cdot}B, C\}_n$
- $\lambda_n^{\text{ss-out}} = \{X, B, C^{\text{out}}\}_n$
- $\lambda_n^{\text{ss-in}} = \{A^{\text{toe}}, C^{\text{loop}}\}_n$
- $\lambda_n^{\text{reactive}} = \{A{\cdot}B, C, X, B, C^{\text{out}}, A^{\text{toe}}, C^{\text{loop}}\}_n$
- $\lambda_n^{\text{cognate}} = \{X{\cdot}A, B{\cdot}C, X{\cdot}A^{\text{toe}}, B{\cdot}C^{\text{loop}}\}_n$

based on the definitions (listed 5′ to 3′ using the sequence domain notation of Figure B.5):

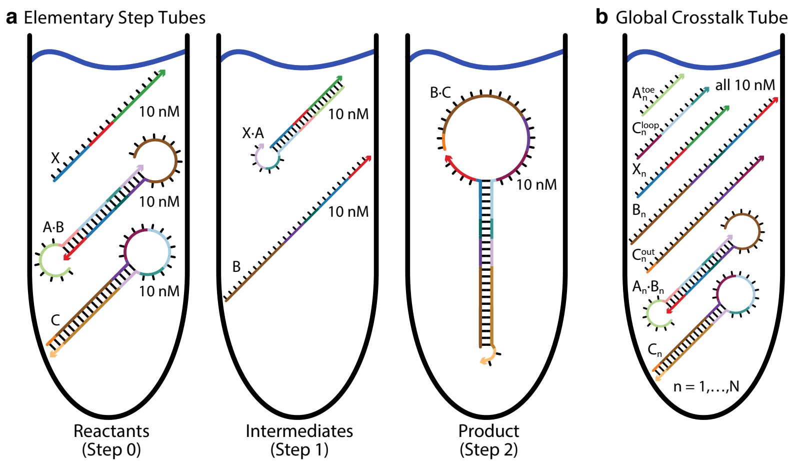
- A ≡ c*-b*-a*-z*-y*
- A^{toe} ≡ c*
- B ≡ x-y-z-a-b
- C ≡ C^{out}-C^{in}
- C^{loop} ≡ s-a*-z*
- C^{in} ≡ a*-z*-y*-x*-w*
- C^{out} ≡ w-x-y-s
- X ≡ a-b-c

Note: $\mathrm{C}_n^{\mathrm{loop}}$ includes portions of both $\mathrm{C}_n^{\mathrm{in}}$ and $\mathrm{C}_n^{\mathrm{out}}$. Including $\mathrm{C}_n^{\mathrm{loop}}$ in $\lambda_n^{\mathrm{reactive}}$ is not redundant with inclusion of C_n because pairing to the loop would cause a pseudoknot and hence will not be checked by the ensemble except if the interaction opens the hairpin. We want to be able to check nucleation with the loop even when the hairpin remains closed, so we include $\mathrm{C}_n^{\mathrm{loop}}$ in $\lambda_n^{\mathrm{reactive}}$.

To illustrate this approach, Figure 3.2a depicts target test tubes for the reaction pathway of Figure 3.1. There are three elementary step tubes, each containing on-target complexes corresponding to the products of the corresponding step: the Reactants tube (Step 0) contains on-targets X, A·B, and C; the Step 1 tube contains on-targets X·A and B; the Step 2 tube contains on-target B·C. Within each target test tube, each on-target complex is depicted by its target secondary structure labeled with its target concentration. Each elementary step tube also contains off-targets (with vanishing target concentration) of two varieties: reactants that are intended to fully convert into the on-pathway products, and off-pathway crosstalk products between these same reactants. Hence, these elementary step tubes design for full conversion of cognate reactants into cognate products and against local crosstalk between these same reactants.

To simultaneously design N orthogonal systems, three elementary step tubes of the type shown in Figure 3.2a are specified for each system. Furthermore, to design against off-pathway interactions within and between systems, a single global crosstalk tube is specified in Figure 3.2b. In the global crosstalk tube, the on-target complexes correspond to all reactive species generated during all elementary steps ($m = 0, 1, 2$) for all systems ($n = 1, \ldots, N$); the off-target complexes correspond to non-cognate interactions between these reactive species (see Section 3.2.3 for details on defining reactive species for a given reaction pathway). Crucially, the global crosstalk tube ensemble omits the cognate products that the reactive species are intended to form (they appear as neither on-targets nor off-targets). Hence, all reactive species in the global crosstalk tube are forced to perform either no reaction (remaining as desired on-targets) or to undergo a crosstalk reaction (forming undesired off-targets), providing the basis for minimization of global crosstalk during sequence optimization. To design 8 orthogonal systems for this reaction pathway, the total number of target test tubes is then $|\Omega| = 3 * 8 + 1 = 25$.

These computational test tube ensembles have two conceptually interesting and practically significant properties. First, each target test tube isolates a different subset of the system components in local equilibrium, enabling optimization of kinetically significant states that would appear insignificant if all components were allowed to interact in a single ensemble. For example, the Step 1 tube of Figure 3.2a simultaneously optimizes for high-yield production of unstructured intermediate B and against appreciable formation of off-target dimer B·B, promoting rapid nucleation of the unstructured toehold in B with the loop of hairpin C during the next step of the reaction pathway. Second, for a tube containing a given set of system components, the cognate products of their interactions can be excluded from the ensemble (appearing as neither on-targets nor off-targets), enabling optimization for high-yield well-structured reactants and against crosstalk. For example, the Reactants tube of Figure 3.2a excludes the cognate product of Step 1 from the ensemble in order to optimize formation of initial reactants X, A·B, and C and discourage competing crosstalk interactions (e.g., X·X, A·A, X·C).



Tube	On-targets (Ψ_h^{on})	Off-targets (Ψ_h^{off})
Step 0_n	$\{\text{X, A·B, C}\}_n$	$\{\text{A, B·C}\}_n \cup \Psi_{0_n}^{L \le L_{\max}} - \{\text{X·A}\}$
Step 1_n	$\{\text{X·A, B}\}_n$	$\{\text{X, A·B}\}_n \cup \Psi_{1_n}^{L \le L_{\max}}$
Step 2_n	$\{\text{B·C}\}_n$	$\{\text{B, C}\}_n \cup \Psi_{2_n}^{L \le L_{\max}}$
Crosstalk	$\cup_{n=1,\ldots,N}\{\lambda_n^{\text{reactive}}\}$	$\Psi_{\text{global}}^{L \le L_{\max}} - \cup_{n=1,\ldots,N}\{\lambda_n^{\text{cognate}}\}$

Figure 3.2: Target test tubes for conditional Dicer substrate formation via shape and sequence transduction with scRNAs (reaction pathway of Figure B.5). Top: Target test tube schematics. Bottom: Target test tube details. Each target test tube contains the depicted on-target complexes (each with the depicted target structure and a target concentration of 10 nM) and the off-target complexes listed in the table (each with vanishing target concentration). To simultaneously design N orthogonal systems, the total number of target test tubes is $|\Omega| = 3N + 1$. $L_{\max} = 2$ for all tubes. Design conditions: RNA in 1 M Na$^+$ at 37 °C.

3.2.4 Design Objective Function

To provide a physically meaningful objective function for optimizing the equilibrium base-pairing properties of a single test tube of interacting nucleic acid strands, we previously derived the *test tube ensemble defect*[17], C_h, quantifying the equilibrium concentration of incorrectly paired nucleotides over the ensemble of test tube h. Let

$$\mathcal{M}_h \equiv C_h / y_h^{\text{nt}} \in (0, 1) \tag{3.1}$$

denote the equilibrium fraction of incorrectly paired nucleotides in tube h. Here,

$$y_h^{\text{nt}} \equiv \sum_{j \in \Psi_h^{\text{on}}} |\phi_j| y_{h,j}$$

is the total concentration of nucleotides in tube h, where ϕ_j denotes the sequence of complex j. As $\mathcal{M}_h$ approaches zero, each on-target complex, $j \in \Psi_h^{\text{on}}$, approaches its target concentration, $y_{h,j}$, and is dominated by its target structure, s_j, and each off-target complex, $j \in \Psi_h^{\text{off}}$, forms with vanishing target concentration.

Generalizing to the multistate test tube ensemble, the *multistate test tube ensemble defect*,

$$\mathcal{M} \equiv \frac{1}{|\Omega|} \sum_{h \in \Omega} \mathcal{M}_h \quad \in (0, 1) \tag{3.2}$$

quantifies the average equilibrium fraction of incorrectly paired nucleotides over the test tubes $h \in \Omega$. The goal is to design a set of sequences such that the multistate test tube ensemble defect, $\mathcal{M}$, satisfies the *stop condition*,

$$\mathcal{M} \leq f_{\text{stop}}, \tag{3.3}$$

for a user-specified value of $f_{\text{stop}} \in (0, 1)$.

In some cases, the user may wish to alter the relative weighting of defect contributions within $\mathcal{M}$ to prioritize or deprioritize design quality for a portion of the design ensemble. To provide flexibility, we permit the user to define custom defect weights for the contribution of each nucleotide, complex, and test tube to $\mathcal{M}$ (see Section B.1.6). With the default value of unity for all weights, this objective function is simply the multistate test tube ensemble defect (3.2). With custom weights, the physical meaning of the objective function is distorted in the service of adjusting design priorities.

Table 3.1: Sequence Constraints.

Constraint type	Constraint relation*	Nucleotides
Assignment	$(\phi^a) \in R_a^{\text{assignment}} \equiv \{(q^1)\}$	1
Match	$(\phi^a, \phi^b) \in R_{a,b}^{\text{match}} \equiv \{(\mathtt{A},\mathtt{A}), (\mathtt{C},\mathtt{C}), (\mathtt{G},\mathtt{G}), (\mathtt{U},\mathtt{U})\}$	2
Watson–Crick	$(\phi^a, \phi^b) \in R_{a,b}^{\text{WC}} \equiv \{(\mathtt{A},\mathtt{U}), (\mathtt{C},\mathtt{G}), (\mathtt{G},\mathtt{C}), (\mathtt{U},\mathtt{A})\}$	2
Complementarity	$(\phi^a, \phi^b) \in R_{a,b}^{\text{complement}} \equiv \{(\mathtt{A},\mathtt{U}), (\mathtt{C},\mathtt{G}), (\mathtt{G},\mathtt{C}), (\mathtt{U},\mathtt{A}), (\mathtt{G},\mathtt{U}), (\mathtt{U},\mathtt{G})\}$	2
Composition	$(\phi^a, \ldots, \phi^b) \in R_{a,\ldots,b}^{\text{composition}} \equiv \{(\phi^a, \ldots, \phi^b) \mid f^{\min} \le \sum_{i=a,\ldots,b} d(\phi^i, q^1)/n \le f^{\max}\}$	$b - a + 1 = n$
Similarity	$(\phi^a, \ldots, \phi^b) \in R_{a,\ldots,b}^{\text{similarity}} \equiv \{(\phi^a, \ldots, \phi^b) \mid f^{\min} \le d((\phi^a, \ldots, \phi^b), (q^1, \ldots, q^n))/n \le f^{\max}\}$	$b - a + 1 = n$
Pattern prevention	$(\phi^a, \ldots, \phi^b) \in R_{a,\ldots,b}^{\text{pattern}} \equiv \{(\phi^a, \ldots, \phi^b) \mid (q^1, \ldots, q^n)$ is not a subsequence of $(\phi^a, \ldots, \phi^b)\}$	$b - a + 1 \ge n$
Library	$(\phi^a, \ldots, \phi^b) \in R_{a,\ldots,b}^{\text{library}} \equiv \{(q_1^1, \ldots, q_1^n), \ldots, (q_m^1, \ldots, q_m^n)\}$	$b - a + 1 = n$
Window	$(\phi^a, \ldots, \phi^b) \in R_{a,\ldots,b}^{\text{window}} \equiv \{(\phi^a, \ldots, \phi^b) \mid (\phi^a, \ldots, \phi^b)$ is a subsequence of $(q^1, \ldots, q^n)\}$	$b - a + 1 \le n$

*For user-specified $q^i \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}, \mathtt{M}, \mathtt{R}, \mathtt{W}, \mathtt{S}, \mathtt{Y}, \mathtt{K}, \mathtt{V}, \mathtt{H}, \mathtt{D}, \mathtt{B}, \mathtt{N}\}$.

3.2.5 Sequence Constraints

A nucleic acid reaction pathway imposes sequence constraints on its reactants (e.g., the complementary sequence domains 'a' and 'a*' in the reaction pathway of Figure 3.1). Furthermore, a molecular engineer may wish to impose a variety of additional sequence constraints (e.g., constraining GC content to optimize synthesis, or constraining input X, comprising sequence domains 'a-b-c' in Figure 3.1, to be a subsequence of a particular mRNA).

We provide a unified and extensible framework for imposing diverse types of sequence constraints on the design problem:

- *Assignment Constraint.* Nucleotide a is constrained to have a specified sequence (e.g., A, C, G, U or any of the IUPAC degenerate nucleotide codes; see Table B.1).
- *Match Constraint.* Two nucleotides a and b are constrained to be identical (e.g., if a strand species appears in more than one on-target complex, corresponding nucleotides are constrained to have the same sequence in all complexes).
- *Watson–Crick Constraint.* Two nucleotides a and b are constrained to be Watson-Crick complements (by default, Watson–Crick constraints are implied for all base pairs present in on-target structures).
- *Complementarity Constraint.* Two nucleotides a and b are constrained to be Watson–Crick or wobble complements.
- *Composition Constraint.* Consecutive nucleotides $a, \dots, b$ are constrained to have a sequence composition in a specified range (e.g., a desired GC content can be achieved by constraining the fraction of S nucleotides to fall in the range $[f^{\min}, f^{\max}]$).
- *Similarity Constraint.* Consecutive nucleotides $a, \dots, b$ are constrained to be similar to a specified sequence of length $n = b - a + 1$ to a specified degree (e.g., the fraction of nucleotides matching an mRNA sequence can be constrained to fall in the range $[f^{\min}, f^{\max}]$).
- *Pattern Prevention Constraint.* Consecutive nucleotides $a, \dots, b$ are constrained not to contain a specified subsequence of length $n \leq b - a + 1$ (e.g., prevention of GGGG, which is prone to forming G-quadruplexes[18] that are not accounted for in nearest-neighbor free energy models super[19,20]).
- *Library Constraint.* Consecutive nucleotides $a, \dots, b$ are constrained to be selected from a specified library of m sequences of length $n = b - a + 1$ (e.g., a library of toehold sequences or a library of codons).
- *Window Constraint.* Consecutive nucleotides $a, \dots, b$ are constrained to be a subsequence of a specified source sequence of length $n \geq b - a + 1$ (e.g., the source sequence is an mRNA), or more generally, a subsequence of one of multiple specified source sequences.

Table 3.2: Reaction pathway engineering case studies.

Reaction pathway	Orthogonal systems	Tubes $\lvert\Omega\rvert$	On-targets $\lvert\Psi^{\mathrm{on}}\rvert$	Off-targets $\lvert\Psi^{\mathrm{off}}\rvert$
Conditional self-assembly via	1	5	7	9
hybridization chain reaction (HCR)	8	33	56	520
Boolean logic AND using	1	5	14	52
toehold sequestration gates	8	33	112	3216
Self-assembly of a 3-arm junction via	1	6	11	27
catalytic hairpin assembly (CHA)	8	41	88	1588
Boolean logic AND using	1	3	9	51
a cooperative hybridization gate	8	17	72	2676
Conditional Dicer substrate formation	1	4	9	26
via shape and sequence transduction	8	25	72	1580

Within this framework, each constraint is expressed as a constraint relation (Table 3.1). For some constraint relations, it is convenient to make use of the sequence distance function,

$$d(\phi, q) \equiv \sum_{a \in 1,\ldots,|\phi|} \begin{cases} 0 : \phi^a \in q^a \\ 1 : \phi^a \notin q^a \end{cases},$$

between sequence ϕ and the constraint sequence q of equal length, which may contain degenerate IUPAC nucleotide codes (see Table B.1). For example, $d(\mathtt{ACGU}, \mathtt{SSWW}) = 2$. Additional types of sequence constraints can be supported by specifying new constraint relations.

3.2.6 Constrained Multistate Test Tube Design Problem

Let $\phi_\Psi \equiv \phi_j \; \forall j \in \Psi$ denote the set of sequences for the complexes in Ψ and let $\mathcal{R}$ denote the set of user-specified sequence constraints. To design a set of sequences, Φ_Ψ, for a given a nucleic acid reaction pathway, we specify on-target and off-target complexes within the set of target test tubes, Ω, to represent on-pathway elementary steps and off-pathway crosstalk. The constrained multistate test tube design problem is then:

$$\min_{\phi_\Psi} \mathcal{M} \quad \text{subject to} \quad \mathcal{R}, \tag{3.4}$$

where $\mathcal{M}$ is the multistate test tube ensemble defect over Ω. The sequence design algorithm seeks to iteratively reduce $\mathcal{M}$ while satisfying the constraints in $\mathcal{R}$, terminating sequence optimization upon satisfaction of the stop condition (3.3).

3.3 Methods

3.3.1 Algorithm Overview

Our constrained multistate test tube design algorithm generalizes the test tube design algorithm of Wolfe and Pierce[17] to perform sequence design over an ensemble of an arbitrary number of target test

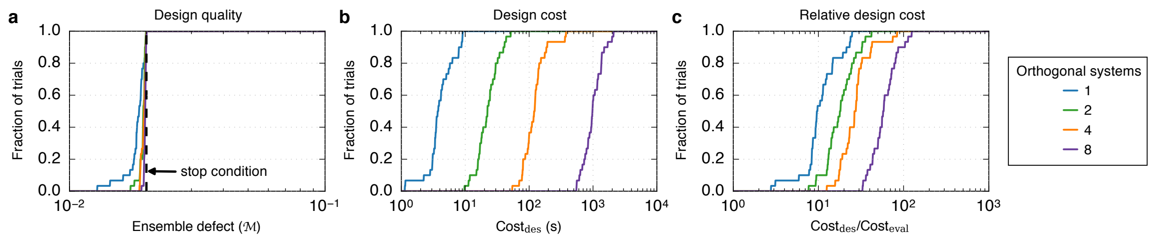


Figure 3.3: Algorithm performance for simultaneous design of $N = 1, 2, 4$, or 8 orthogonal systems using the target test tubes of Figure 3.2. (a) Design quality. The stop condition is depicted as a dashed black line. (b) Design cost. (c) Cost of sequence design relative to a single evaluation of the objective function. Case study: conditional Dicer substrate formation via shape and sequence transduction with scRNAs. See Figure B.11 for comparison of all case studies.

tubes subject to diverse user-specified sequence constraints. The underlying physical model is based on nucleic acid secondary structure and nearest-neighbor free energy parameters (Section B.1.1). The objective function, $\mathcal{M}$, is reduced via iterative mutation of a random initial sequence. Because of the high computational cost of calculating the objective function (Section B.1.2), it is important to avoid direct recalculation of $\mathcal{M}$ in evaluating each candidate mutation. We exploit three concepts to enable efficient calculation of the objective function estimate, $\tilde{\mathcal{M}}$: using *test tube ensemble focusing*, sequence optimization initially focuses on only the on-target portion of each test tube ensemble (Section B.1.3); using *hierarchical ensemble decomposition*, the structural ensemble of each on-target complex is hierarchically decomposed into a tree of conditional subensembles, yielding a forest of decomposition trees (Section B.1.4); by calculating *conditional physical properties* over the conditional structural ensembles at any level within the decomposition forest, it is possible to efficiently estimate the equilibrium base-pairing properties of the multistate test tube ensemble (Section B.1.5). Optional *defect weights* enable the user to adjust design priorities within this ensemble (Section B.1.6). To minimize computational cost, candidate mutations are evaluated at the leaf level of the decomposition forest (Section B.1.7). As optimized subsequences are merged toward the root level of the forest, any emergent defects are eliminated via ensemble redecomposition from the parent level on down and sequence reoptimization from the leaf level on up (Section B.1.8). After subsequences are successfully merged to the root level, the exact objective function, $\mathcal{M}$, is calculated for the first time, explicitly checking for the effect of the previously neglected off-target complexes. Any off-target complexes observed to form at appreciable concentration are hierarchically decomposed, added to the decomposition forest, and actively destabilized during subsequent forest reoptimization (Section B.1.9). When decomposition or focusing defects are encountered, hierarchical ensemble decomposition is performed using multiple exclusive split-points (Section B.1.10). Throughout the sequence optimization process, whenever the sequence is initialized, mutated, or reseeded, we solve a *constraint satisfaction problem* to obtain valid sequences satisfying all constraints in $\mathcal{R}$ (Section B.1.11). The algorithm flow is detailed in the pseudocode of Algorithm B.1.

3.3.2 Implementation

The constrained multistate test tube design algorithm is coded in the C and C++ programming languages. The algorithm is available for non-commercial academic use as part of the NUPACK web application and source code (`www.nupack.org`)[21].

3.3.3 Sequence Design Trials

For each design problem, 30 independent design trials were performed. Design trials were run on a cluster of 2.53 GHz Intel E5540 Xeon dual-processor/quad-core nodes with 24 GB of memory per node. Each trial was run on one computational core using the default algorithm parameters of Table B.2. Design quality is quantified by the multistate test tube ensemble defect $\mathcal{M}$. To design N orthogonal systems, all nucleotide, complex, and tube weights are left at the default value of 1 except for the global crosstalk tube which is assigned a weight of N to prevent the effect of crosstalk from being diluted in the design objective function as the number of orthogonal systems increases. Data are typically plotted[22] as cumulative histograms over design trials. Relative design cost is quantified by dividing the cost of sequence design (cost_{des}) by the cost of a single evaluation of the multistate test tube ensemble defect ($\text{cost}_{\text{eval}}$). Designs are performed in 1M Na^+ at 37 °C for RNA (conditional Dicer substrate formation case study) and 25 °C for DNA (all other case studies). For each design trial, the stop condition is $f_{\text{stop}} = 0.02$ (i.e., no more than 2% of nucleotides incorrectly paired at equilibrium over the multistate test tube ensemble, Ω).

3.4 Results

3.4.1 Reaction Pathway Engineering Case Studies

To examine algorithm performance, we consider a selection of reaction pathways from the molecular programming literature (Section B.2.1):

- *Conditional self-assembly via hybridization chain reaction (HCR).* A single-stranded input X triggers self-assembly of metastable hairpins into a nicked double-stranded polymer[4].
- *Boolean logic AND using toehold sequestration gates.* Detection of two independent single-stranded inputs X and Y triggers release of a single-stranded output[5].
- *Self-assembly of a 3-arm junction via catalytic hairpin assembly (CHA).* A single-stranded input X catalyzes self-assembly of a 3-arm branched junction from metastable hairpins[8].
- *Boolean logic AND using a cooperative hybridization gate.* Two independent single-stranded inputs X and Y cooperatively displace a single-stranded output[16].
- *Conditional Dicer substrate formation via shape and sequence transduction.* Detection of a single-stranded input X leads to formation of a double-stranded Dicer substrate targeting an independent output sequence Y for silencing[12].

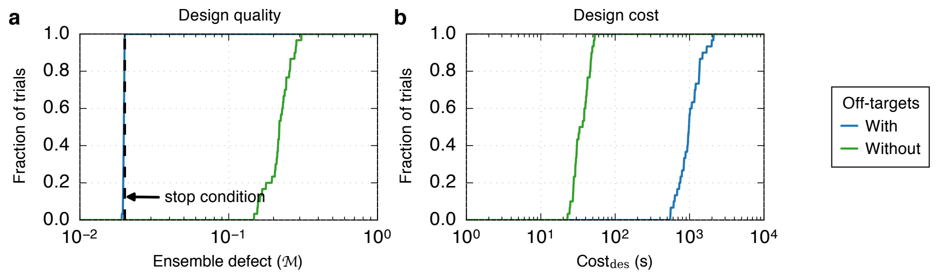


Figure 3.4: Importance of negative design in reducing crosstalk. Comparison of designs performed with or without off-targets in the design ensemble. (a) Design quality evaluated by calculating the multistate test tube ensemble defect ($\mathcal{M}$) over the ensemble containing off-targets. The stop condition is depicted as a dashed black line. (b) Design cost. Case study: conditional Dicer substrate formation via shape and sequence transduction with scRNAs (N = 8 orthogonal systems). See Figure B.18 for comparison of all case studies.

For each reaction pathway, we define a set of target test tubes specifying 1, 2, 4, or 8 orthogonal systems intended to operate independently in the same sample (Section 3.2.3). These constrained multistate test tube design problems involve up to dozens of test tubes, dozens of on-target complexes, and thousands of off-target complexes (Table 3.2).

3.4.2 Algorithm Performance for Constrained Multistate Test Tube Design

Figure 3.3 demonstrates the performance of the constrained multistate test tube design algorithm for the target test tubes of Figure 3.2, corresponding to conditional Dicer substrate formation via shape and sequence transduction with scRNAs. For each target test tube, the algorithm designs for the depicted on-target complexes (each with a target secondary structure and target concentration) and against all off-target complexes. The size of the design ensemble ranges from 4 tubes containing a total of 9 on-target and 26 off-target complexes for design of 1 system to 25 tubes containing a total of 72 on-target and 1580 off-target complexes for design of 8 orthogonal systems (Table 3.2). Sequences are designed subject only to implicit sequence constraints inherent to the sequence domain specification for each strand (e.g., match constraints for domain “a" appearing in multiple complexes, Watson-Crick constraints for complementary domains “a" and “a*"). Typical design trials achieve the desired design quality (stop condition $\mathcal{M} \leq 0.02$; panel a) and typical design costs range from seconds for the design of 1 system to minutes for simultaneous design of 8 orthogonal systems (panel b). The typical cost of design relative to the cost of analysis ranges from approximately 10 to 60 as the number of orthogonal systems increases from 1 to 8 (panel c). These rising relative design costs reflect the increasing challenge of designing against crosstalk as the number of orthogonal systems increases.

For the five engineering case studies of Table 3.2, typical design trials achieve the 2% stop condition (Figure B.11). For simultaneous design of 8 orthogonal systems, the typical cost of design relative to the cost of analysis ranges from a factor of 60 to 1300. If desired, the design cost can be reduced by relaxing the design quality requirements; using $f_{\text{stop}} = 0.05$ instead of 0.02, the desired design quality is typically achieved with a relative design cost ranging from a factor of 30 to 300 (Figure B.12).

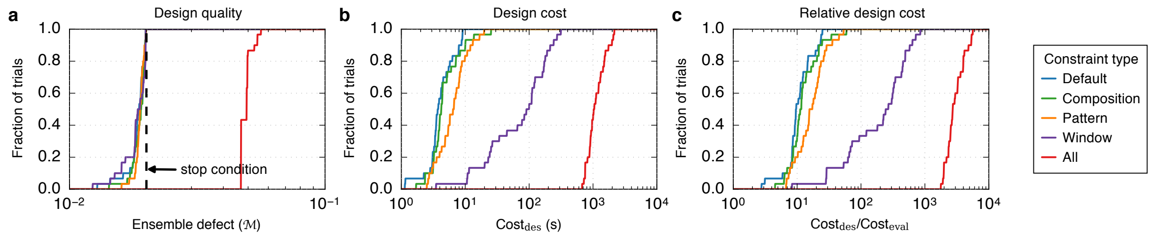


Figure 3.5: Algorithm performance including explicit sequence constraints. Default: implicit sequence constraints inherent to the reaction pathway (these constraints are also present in the other cases that follow). Composition constraint: fraction of $\mathtt{S} \in [0.45, 0.55]$. Pattern constraint: prevent {AAAA, CCCC, GGGG, UUUU, KKKKKK, MMMMMM, RRRRRR, SSSSSS, WWWWWW, YYYYYY}. Window constraints: input X and output Y constrained to be subsequences of two different mRNAs (i.e., biological sequence constraints; see Section B.2.5). All: all of the above constraints. (a) Design quality. The stop condition is depicted as a dashed black line. (b) Design cost. (c) Cost of sequence design relative to a single evaluation of the objective function. Case study: conditional Dicer substrate formation via shape and sequence transduction with scRNAs ($N = 1$ system). See Figure B.19 for comparison of all case studies.

At the conclusion of sequence design, the distribution of residual defects across target test tubes and across complexes within each target test tube depends on the idiosyncrasies of each case study (see Section B.2.3), with the more challenging tubes and complexes retaining higher defects. If desired, nucleotide, complex, or tube defect weights can be adjusted from their default values of unity to prioritize or deprioritize reduction of the corresponding defect contributions to the design objective function, $\mathcal{M}$.

3.4.3 Importance of Negative Design in Reducing Crosstalk

The target test tubes summarized in Table 3.2 and detailed in Section 3.2.3 contain both on-target and off-target complexes, implementing both a positive design paradigm (designing for the target concentration of on-target complexes) and a negative design paradigm (designing against the formation of off-target complexes). Is it important to include off-targets in the design ensemble and explicitly destabilize these off-pathway interactions in order to arrive at sequence designs with low crosstalk? To examine this question, Figure 3.4 re-examines the simultaneous design of 8 orthogonal systems using either the full design ensemble (a total of 72 on-target and 1580 off-target complexes) or a reduced design ensemble that omits all off-target complexes, evaluating the quality of the resulting designs over the full ensemble. If the design ensemble contains no off-targets, the typical ensemble defect of the final sequence designs increases by an order of magnitude (from 2% to over 20%), emphasizing the importance of explicitly destabilizing off-targets. The other engineering case studies similarly emphasize the importance of negative design in reducing crosstalk (Figure B.18).

3.4.4 Effect of Sequence Constraints

Figure 3.5 illustrates the effects of imposing explicit sequence constraints (composition: constraining GC content, pattern: preventing 4-nt stretches of any one nucleotide and 6-nt stretches of any two nucleotides, window: constraining input X and output Y to be subsequences of different mRNAs) on the design of a single system. Using any of these constraint types alone, typical design trials satisfy the 2% stop condition (panel a). For composition and pattern constraints, the cost of design is

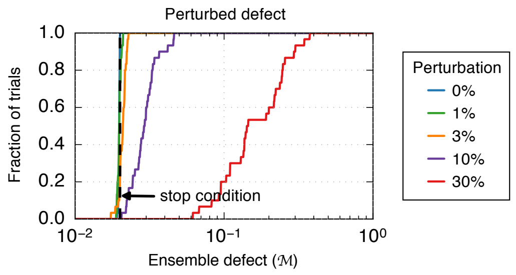


Figure 3.6: Robustness of design quality assessments to perturbations in model parameters. For each design trial, the median multistate test tube ensemble defect was calculated over 100 perturbed physical models (each parameter perturbed by Gaussian noise with a standard deviation of 0, 1, 3, 10, or 30% of the parameter modulus). The stop condition is depicted as a dashed black line. Case study: conditional Dicer substrate formation via shape and sequence transduction with scRNAs ($N = 8$ orthogonal systems). See Figure B.20 for comparison of all case studies.

approximately one order of magnitude higher than the cost of analysis (panel c), while using window constraints to impose biological compatibility, the relative design cost increases to two orders of magnitude. Imposing all three constraint types simultaneously, the algorithm is typically unable to reduce the multistate ensemble defect below 5%, and the relative design cost increases to three orders of magnitude. It is noteworthy that imposing biological sequence constraints dramatically reduces the size of the design space: for the 39 nucleotides constrained by mRNAs X and Y, the number of feasible candidate sequences decreases from 3×10^{23} to 4×10^{6}; additionally imposing composition and pattern constraints decreases this number to 3×10^{4} (see Section B.2.5). This dramatic reduction in sequence space increases both the challenge of jumping between feasible candidate sequences during the search process and the challenge of achieving a low multistate ensemble defect for the final design. Sequence-constrained versions of the other engineering case studies reveal similar trends (Figure B.19), illustrating that sequence constraints increase the degree of difficulty both in terms of design quality and design cost, and emphasizing the desirability of constraining the design space only to the extent necessary.

3.4.5 Robustness of Predictions to Model Perturbations

As the empirical parameter sets underlying nucleic acid secondary structure models[19,20,23–25] are further refined going forward, it is important that the predicted design quality is robust to perturbations in the parameters. Figure 3.6 demonstrates that the predicted quality is typically robust to 3% parameter perturbations (with typical ensemble defect comparable to the 2% stop condition), and even to 10% parameter perturbations (with the typical defect increasing to only 3%), but not to 30% parameter perturbations (with the typical defect increasing to 14%). Qualitatively similar trends are seen for the other case studies (Figure B.20).

3.5 Discussion

To enable sequence design for reaction pathway engineering, the *multistate test tube design* problem builds on three subsidiary design problems: *complex design*, *multistate complex design*, and *test tube design*. For each subsidiary design problem, we describe the design ensemble, define an

Table 3.3: Nucleic acid sequence design ensembles.

		Per test tube $h \in \Omega$	
Design problem	Test tubes $\lvert\Omega\rvert$	On-target complexes $\lvert\Psi_h^{\text{on}}\rvert$	Off-target complexes $\lvert\Psi_h^{\text{off}}\rvert$
Complex design	1	1	0
Multistate complex design	Arbitrary	1	0
Test tube design	1	Arbitrary	Arbitrary
Multistate test tube design	Arbitrary	Arbitrary	Arbitrary

ensemble defect that provides a physically meaningful objective function for optimizing sequence quality over this ensemble, and examine the extent to which the resulting formulation implements a positive design paradigm (stabilize on-targets) and a negative design paradigm (destabilize off-targets). These comparisons illuminate the conceptual properties of the current formulation and the manner in which it extends existing design capabilities.

3.5.1 Complex Design

For complex design, the goal is to design the equilibrium base pairing properties of a complex of (one or more) interacting nucleic acid strands. This subsidiary design problem is the foundation on which the other three design problems build, and has attracted the most algorithm development to date[26–43].

Design Ensemble. For complex design, the user specifies an on-target complex, j, with on-target secondary structure, s_j. We may view complex design as a special case of multistate test tube design where the design ensemble comprises a single target test tube containing a single on-target complex and no off-target complexes (Table 3.3).

Design Objective Function. The *complex ensemble defect*,

$$n(\phi_j, s_j) = |\phi_j| - \sum_{\substack{1\le a\le|\phi_j|\\ 1\le b\le|\phi_j|+1}} P^{a,b}(\phi_j)S^{a,b}(s_j),$$

quantifies the equilibrium number of incorrectly paired nucleotides over the ensemble of complex j[29,36]. Here, $P(\phi_j)$ is the equilibrium base-pairing probability matrix and $S(s_j)$ is the target structure matrix for complex j (see Section B.1.2). Let

$$\mathcal{N}_j \equiv n(\phi_j, s_j)/|\phi_j| \quad \in (0,1)$$

denote the equilibrium fraction of incorrectly paired nucleotides in the ensemble of complex j. The goal is to design a sequence, ϕ_j, that satisfies the stop condition,

$$\mathcal{N}_j \le f_{\text{stop}},$$

for a user-specified value of $f_{\text{stop}} \in (0, 1)$.

Design Paradigms. Optimization of the complex ensemble defect implements a positive design paradigm (designing for the on-target structure) and a negative design paradigm (designing against all off-target structures) within the ensemble of the on-target complex[29,36]. Both paradigms are crucial to achieve high-quality sequence designs with low complex ensemble defect[29,36]. Because the complex design ensemble is equivalent to a target test tube containing a single on-target complex and no off-target complexes, complex design implements only a positive design paradigm (designing for the on-target complex) within the ensemble of the target test tube (Table 3.4).

3.5.2 Multistate Complex Design

The complex design ensemble comprises a single on-target complex, and hence does not address the multistate challenges inherent in reaction pathway engineering. To address this need, multistate complex design expands the design ensemble to include multiple on-target complexes[44]. For multistate complex design, the goal is to engineer the equilibrium base pairing properties of an arbitrary number of on-target complexes, each representing a reactant, intermediate, or product state along the reaction pathway.

Design Ensemble. For multistate complex design, the user specifies a set of on-target complexes, Ψ. For each on-target complex, $j \in \Psi$, the user specifies a target secondary structure, s_j[44]. We may view multistate complex design as a special case of multistate test tube design where the design ensemble contains an arbitrary number of target test tubes each containing a single on-target complex and no off-target complexes (Table 3.3).

Design Objective Function. The *multistate complex ensemble defect*,

$$\mathcal{N} \equiv \frac{1}{|\Psi|} \sum_{j \in \Psi} \mathcal{N}_j \quad \in (0, 1),$$

quantifies the average equilibrium fraction of incorrectly paired nucleotides over the complexes $j \in \Psi$. The goal is to design a set of sequences, ϕ_Ψ, that satisfy the stop condition

$$\mathcal{N} \le f_{\text{stop}},$$

for a user-specified value of $f_{\text{stop}} \in (0, 1)$.

Design Paradigms. Multistate complex design inherits the benefits and shortcomings of complex design for each reactant, intermediate, or product state along the reaction pathway, implementing positive and negative design paradigms within the ensemble of each on-target complex, but only a positive design paradigm within the ensemble of each target test tube (Table 3.4).

3.5.3 Test Tube Design

With complex design and multistate complex design, neither the concentration of the desired on-target complex, nor the concentrations of undesired off-target complexes are considered. As a result,

Table 3.4: Nucleic acid sequence design paradigms.

	Paradigms within complex		Paradigms within test tube		
Ensemble defect	Positive	Negative	Positive	Negative	States
Complex	✓	✓	✓		1
Multistate complex	✓	✓	✓		Arbitrary
Test tube	✓	✓	✓	✓	1
Multistate test tube	✓	✓	✓	✓	Arbitrary

sequences that are successfully optimized to stabilize a target secondary structure in the context of an on-target complex, may nonetheless fail to ensure that this complex forms at appreciable concentration when the strands are introduced into a test tube[17]. To address this critical shortcoming, test tube design expands the complex design ensemble to include off-target complexes[17]. For test tube design, the goal is to engineer the equilibrium base-pairing properties of a test tube of interacting nucleic acid strands.

Design Ensemble. For test tube design[17], the user specifies a target test tube h containing a set of on-target complexes, Ψ_h^{on}, and a set of off-target complexes, Ψ_h^{off}. For each on-target complex, $j \in \Psi_h^{\text{on}}$, the user specifies a target secondary structure, s_j, and a target concentration, $y_{h,j}$. For each off-target complex, $j \in \Psi_h^{\text{off}}$, the target concentration is vanishing ($y_{h,j} = 0$) and there is no target structure ($s_j = \emptyset$). We may view test tube design as a special case of multistate test tube design where the design ensemble contains a single target test tube containing arbitrary numbers of on- and off-target complexes (Table 3.3).

Design Objective Function. The *test tube ensemble defect*,

$$C(\phi_{\Psi_h}, s_{\Psi_h}, y_{h,\Psi_h}) = \sum_{j \in \Psi_h^{\text{on}}} \Big[n(\phi_j, s_j) \min(x_{h,j}, y_{h,j}) + |\phi_j| \max(y_{h,j} - x_{h,j}, 0) \Big] \tag{3.5}$$

quantifies the equilibrium concentration of incorrectly paired nucleotides over the ensemble of test tube h[17]. Here, $x_{h,j}$ is the equilibrium concentration of complex j in tube h (see Section B.1.2). For each on-target complex, $j \in \Psi_h^{\text{on}}$, the first term in the sum represents the *structural defect*, quantifying the concentration of nucleotides that are in an incorrect base-pairing state within the ensemble of complex j, and the second term in the sum represents the *concentration defect*, quantifying the concentration of nucleotides that are in an incorrect base-pairing state because there is a deficiency in the concentration of complex j. For each off-target complex, $j \in \Psi_h^{\text{off}}$, the structural and concentration defects are identically zero, since $y_{h,j} = 0$. This does not mean that the defects associated with off-targets are ignored. By conservation of mass, non-zero off-target concentrations imply deficiencies in on-target concentrations, and these concentration defects are quantified by (3.5)[17]. Using (3.1) to define the normalized test tube ensemble defect, $\mathcal{M}_h$, in terms of $C_h \equiv$

$C(\phi_{\Psi_h}, s_{\Psi_h}, y_{h,\Psi_h})$, the goal is to design a set of sequences, ϕ_{Ψ_h}, that satisfy the stop condition,

$$\mathcal{M}_h \leq f_{\text{stop}},$$

for a user-specified value of $f_{\text{stop}} \in (0, 1)$.

Design Paradigms. Optimization of the test tube ensemble defect implements a positive design paradigm and a negative design paradigm at two levels (Table 3.4): 1) within the ensemble of each on-target complex (designing for the on-target structure and against all off-target structures)[29,36], and 2) within the ensemble of the target test tube (designing for the target concentration of each on-target complex and against the formation of all off-target complexes)[17]. Both paradigms are crucial at both levels in order to achieve high-quality sequence designs with low test tube ensemble defect[17,29,36].

3.5.4 Multistate Test Tube Design

The present work extends the conceptual benefits of test tube design to address the multistate demands of reaction pathway engineering.

Design Ensemble. The multistate test tube design ensemble generalizes each of the three subsidiary design ensembles, encompassing an arbitrary number of target test tubes, each containing arbitrary numbers of on- and off-target complexes (Table 3.3).

Design Objective Function. Likewise, the multistate test tube ensemble defect, $\mathcal{M}$, generalizes each of the three subsidiary ensemble defects. When the design ensemble comprises a single target test tube containing a single on-target complex and no off-target complexes, $\mathcal{M}$ reduces to $\mathcal{N}_j$, the normalized complex ensemble defect[29,36]. When the design ensemble comprises multiple target test tubes, each containing a single on-target complex and no off-target complexes, $\mathcal{M}$ reduces to $\mathcal{N}$, the multistate complex ensemble defect. When the design ensemble comprises a single target test tube containing arbitrary numbers of on- and off-target complexes, $\mathcal{M}$ reduces to $\mathcal{M}_h$, the normalized test tube ensemble defect[17].

Design Paradigms. Multistate test tube design inherits the conceptual benefits of test tube design for each target test tube (reactant, intermediate, product, and global crosstalk), implementing positive and negative design paradigms both within the ensemble of each on-target complex and within the ensemble of each target test tube (Table 3.4).

3.6 Conclusions

Summary. Constrained multistate test tube design enables sequence design for nucleic acid reaction pathway engineering. The design ensemble uses a set of target test tubes to represent reactant, intermediate, and product states along the reaction pathway, as well as to model crosstalk between components. Each target test tube contains a set of desired on-target complexes (each with a target secondary structure and target concentration) and a set of undesired off-target complexes (each with vanishing target concentration). Sequence quality is quantified by the ensemble defect, representing the average equilibrium fraction of incorrectly paired nucleotides over the test tubes in the ensemble. Sequence optimization is performed by reducing the ensemble defect subject to user-specified sequence constraints. These sequence constraints can dramatically reduce the size of the design space (e.g., by constraining a sequence domain to be a subsequence of an mRNA). To effectively navigate the available sequence space, each valid candidate sequence is generated by solving a constraint satisfaction problem. The candidate sequence is then accepted or rejected based on an estimate of the ensemble defect calculated efficiently via hierarchical ensemble decomposition[17].

Design Paradigms. Optimization of the ensemble defect implements a positive design paradigm and a negative design paradigm to optimize for on-pathway interactions and against off-pathway interactions at two levels: within the ensemble of each on-target complex (designing for the on-target structure and against all off-target structures)[29,36], and within the ensemble of each target test tube (designing for the target concentration of each on-target complex and against the formation of all off-target complexes)[17]. Both paradigms are critical at both levels in order to robustly optimize ensemble properties over the design ensemble[17,29,36].

NUPACK Software Suite. This work unifies and generalizes the complex design[36], multistate complex design super[44], test tube design[17], and (pre-publication) multistate test tube design tools provided by the NUPACK software suite (`nupack.org`), which have been used to engineer diverse nucleic acid structures, devices, and systems[4,6,8,12,45–72].

Towards a Compiler for Molecular Programming. By enabling systematic reaction pathway engineering, constrained multistate test tube design takes an important step toward our long-term goal of developing a compiler for programming molecular function. Such a compiler would accept as input a reaction pathway specification and produce as output a set of nucleic acid sequences that implement the desired function.

Future Work. To achieve the goal of molecular compilation, additional work is needed to automate the specification of target test tubes given a desired reaction pathway. To engineer synthetic systems that operate in a biological context, where most of the nucleotides are native to a host organism rather than designed by the engineer, it may prove important to actively design against crosstalk with some or all of the native transcriptome, potentially requiring further advances to the sequence design formulation and algorithm. In situations where researchers are designing nucleic acid components intended to interact with proteins or small molecules, it may be desirable to incorporate user-specified energetic penalties or rewards into the design objective function in order to approximate molecular interactions that fall outside the scope of nucleic acid secondary structure models.

3.7 Associated Content

Supporting Information

Algorithm details, pseudocode, default algorithm parameters. Engineering case studies: reaction pathways, specification of target test tubes, algorithm performance, residual defects, importance of negative design in reducing crosstalk, effect of sequence constraints, robustness of predictions to model perturbations. Additional design studies: performance for test tube design, performance for complex design. This material is available free of charge via the Internet at http://pubs.acs.org.

3.8 Author Information

Corresponding Author
E-mail: niles@caltech.edu

Author Contributions
*These authors contributed equally.

Notes
The authors declare competing financial interests in the form of patents and pending patent applications.

3.9 Acknowledgments

This work was funded by the National Science Foundation via the Molecular Programming Project (NSF-CCF-0832824 and NSF-CCF-1317694), by the Gordon and Betty Moore Foundation (GBMF2809), by the Beckman Institute at Caltech (PMTC), by a Christensen Fellowship at St Catherine's College, University of Oxford, by the John Simon Guggenheim Memorial Foundation, by a Professorial Fellowship at Balliol College, University of Oxford, and by the Eastman Visiting Professorship at the University of Oxford.

Bibliography

[1] J. Bath and A. J. Turberfield. “DNA nanomachines”. In: *Nat. Nanotechnol.* 2 (2007), pp. 275–284.

[2] D. Y. Zhang and G. Seelig. “Dynamic DNA Nanotechnology Using Strand-Displacement Reactions”. In: *Nature Chem.* 3.2 (2011), pp. 103–113.

[3] B. Yurke, A. J. Turberfield, A. P. Mills Jr., F. C. Simmel, and J. L. Neumann. “A DNA-Fuelled Molecular Machine Made of DNA”. In: *Nature* 406.6796 (2000), pp. 605–608.

[4] R. M. Dirks and N. A. Pierce. “Triggered Amplification by Hybridization Chain Reaction”. In: *Proc. Natl. Acad. Sci. USA* 101.43 (2004), pp. 15275–15278.

[5] G. Seelig, D. Soloveichik, D. Zhang, and E. Winfree. “Enzyme-free nucleic acid logic circuits”. In: *Science* 314.5805 (2006), pp. 1585–1588.

[6] S. Venkataraman, R. M. Dirks, P. W. K. Rothemund, E. Winfree, and N. A. Pierce. “An autonomous polymerization motor powered by DNA hybridization”. In: *Nat. Nanotechnol.* 2 (2007), pp. 490–494.

[7] D. Zhang, A. Turberfield, B. Yurke, and E. Winfree. “Engineering entropy-driven reactions and networks catalyzed by DNA”. In: *Science* 318.5853 (2007), pp. 1121–1125.

[8] P. Yin, H. M. T. Choi, C. R. Calvert, and N. A. Pierce. “Programming biomolecular self-assembly pathways”. In: *Nature* 451.7176 (2008), pp. 318–322.

[9] D. Zhang and E. Winfree. “Control of DNA strand displacement kinetics using toehold exchange”. In: *J. Am. Chem. Soc.* 131 (2009), pp. 17303–17314.

[10] L. Qian and E. Winfree. “Scaling Up Digital Circuit Computation with DNA Strand Displacement Cascades”. In: *Science* 332.6034 (2011), pp. 1196–1201.

[11] A. Turberfield, J. Mitchell, B. Yurke, A. Mills Jr., M. Blakey, and F. Simmel. “DNA fuel for free-running nanomachines”. In: *Phys. Rev. Lett.* 90.11 (2003), p. 118102.

[12] L. M. Hochrein, M. Schwarzkopf, M. Shahgholi, P. Yin, and N. A. Pierce. “Conditional Dicer substrate formation via shape and sequence transduction with small conditional RNAs”. In: *J. Am. Chem. Soc.* 135.46 (2013), pp. 17322–17330.

[13] J. Bois, S. Venkataraman, H. Choi, A. Spakowitz, Z.-G. Wang, and N. Pierce. “Topological constraints in nucleic acid hybridization kinetics.” In: *Nucleic Acids Res.* 33.13 (2005), pp. 4090–4095.

[14] G. Seelig, B. Yurke, and E. Winfree. “Catalyzed relaxation of a metastable DNA fuel.” In: *J. Am. Chem. Soc.* 128.37 (2006), pp. 12211–12220.

[15] N. L. Dabby. “Synthetic molecular machines for active self-assembly: prototype algorithms, designs, and experimental study”. Ph.D. Thesis. 2013.

[16] D. Y. Zhang. “Cooperative Hybridization of Oligonucleotides”. In: *J. Am. Chem. Soc.* 133.4 (2011), pp. 1077–1086.

[17] B. R. Wolfe and N. A. Pierce. “Nucleic acid sequence design for a test tube of interacting nucleic acid strands”. In: *ACS Synth. Biol.* 4.10 (2015), pp. 1086–1100.

[18] N. Saini, Y. Zhang, K. Usdin, and K. S. Lobachev. “When secondary comes first – The importance of non-canonical DNA structures”. In: *Biochimie* 95.2 (2013), pp. 117–123.

[19] D. H. Mathews, J. Sabina, M. Zuker, and D. H. Turner. “Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure”. In: *J. Mol. Biol.* 288 (1999), pp. 911–940.

[20] J. SantaLucia Jr. and D. Hicks. “The thermodynamics of DNA structural motifs”. In: *Annu. Rev. Biophys. Biomol. Struct.* 33 (2004), pp. 415–440.

[21] J. N. Zadeh, C. D. Steenberg, J. S. Bois, B. R. Wolfe, M. B. Pierce, A. R. Khan, R. M. Dirks, and N. A. Pierce. “NUPACK: Analysis and Design of Nucleic Acid Systems”. In: *J. Comput. Chem.* 32.1 (2011), pp. 170–173.

[22] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Comput. Sci. Eng.* 9.3 (2007), pp. 90–95.

[23] M. J. Serra and D. H. Turner. “Predicting thermodynamic properties of RNA”. In: *Methods Enzymol.* 259 (1995), pp. 242–261.

[24] J. SantaLucia Jr. “A Unified View of Polymer, Dumbbell, and Oligonucleotide DNA Nearest-Neighbor Thermodynamics”. In: *Proc. Natl. Acad. Sci. USA* 95.4 (1998), pp. 1460–1465.

[25] R. T. Koehler and N. Peyret. “Thermodynamic Properties of DNA Sequences: Characteristic Values for the Human Genome”. In: *Bioinformatics* 21.16 (2005), pp. 3333–3339.

[26] I. Hofacker, W. Fontana, P. Stadler, L. Bonhoeffer, M. Tacker, and P. Schuster. “Fast folding and comparison of RNA secondary structures”. In: *Chem. Mon.* 125 (1994), pp. 167–188.

[27] C. Flamm, I. Hofacker, S. Maurer-Stroh, P. Stadler, and M. Zehl. “Design of multistable RNA molecules”. In: *RNA* 7 (2001), pp. 254–265.

[28] R. M. Dirks and N. A. Pierce. “A partition function algorithm for nucleic acid secondary structure including pseudoknots”. In: *J. Comput. Chem.* 24 (2003), pp. 1664–1677.

[29] R. M. Dirks, M. Lin, E. Winfree, and N. A. Pierce. “Paradigms for Computational Nucleic Acid Design”. In: *Nucleic Acids Res.* 32.4 (2004), pp. 1392–1403.

[30] M. Andronescu, A. Fejes, F. Hutter, H. Hoos, and A. Condon. “A new algorithm for RNA secondary structure design”. In: *J. Mol. Biol.* 336.3 (2004), pp. 607–624.

[31] A. Busch and R. Backofen. “INFO-RNA–a fast approach to inverse RNA folding”. In: *Bioinformatics* 22.15 (2006), pp. 1823–1831.

[32] R. Aguirre-Hernandez, H. Hoos, and A. Condon. “Computational RNA secondary structure design: empirical complexity and improved methods”. In: *BMC Bioinformatics* 8 (2007), Article 34.

[33] B. Burghardt and A. Hartmann. “RNA secondary structure design”. In: *Phys. Rev. E* 75 (2007), p. 021920.

[34] J. Z. M. Gao, L. Y. M. Li, and C. M. Reidys. “Inverse folding of RNA pseudoknot structures”. In: *Algorithm Mol. Biol.* 5 (2010), p. 27.

[35] W. J. Shu, M. Liu, H. B. Chen, X. C. Bo, and S. Q. Wang. “ARDesigner: A web-based system for allosteric RNA design”. In: *J. Biotechnol.* 150.4 (2010), pp. 466–473.

[36] J. N. Zadeh, B. R. Wolfe, and N. A. Pierce. “Nucleic Acid Sequence Design via Efficient Ensemble Defect Optimization”. In: *J. Comput. Chem.* 32 (2011), pp. 439–452.

[37] A. Avihoo, A. Churkin, and D. Barash. “RNAexinv: An extended inverse RNA folding from shape and physical attributes to sequences”. In: *BMC Bioinformatics* 12 (2011), p. 319.

[38] E. I. Ramlan and K. P. Zauner. “Design of interacting multi-stable nucleic acids for molecular information processing”. In: *Biosystems* 105.1 (2011), pp. 14–24.

[39] A. Taneda. “MODENA: a multi-objective RNA inverse folding”. In: *Adv. Appl. Bioinforma. Chem.* 4 (2011), pp. 1–12.

[40] A. Levin, M. Lis, Y. Ponty, C. W. O’Donnell, S. Devadas, B. Berger, and J. Waldispuhl. “A global sampling approach to designing and reengineering RNA secondary structures”. In: *Nucleic Acids Res.* 40.20 (2012), pp. 10041–10052.

[41] M. C. Matthies, S. Bienert, and A. E. Torda. “Dynamics in Sequence Space for RNA Secondary Structure Design”. In: *J. Chem. Theory Comput.* 8.10 (2012), pp. 3663–3670.

[42] A. Taneda. “Multi-objective genetic algorithm for pseudoknotted RNA sequence design”. In: *Front. Genet.* 3 (2012), p. 36.

[43] R. B. Lyngso, J. W. J. Anderson, E. Sizikova, A. Badugu, T. Hyland, and J. Hein. “Frnakenstein: multiple target inverse RNA folding”. In: *BMC Bioinformatics* 13 (2012), p. 260.

[44] J. N. Zadeh. “Algorithms for Nucleic Acid Sequence Design”. PhD thesis. 2010.

[45] V. Patzel, S. Rutz, I. Dietrich, C. Köberle, A. Sheffold, and S. Kaufmann. “Design of siRNAs producing unstructured guide-RNAs results in improved RNA interference efficiency”. In: *Nat. Biotechnol.* 23.11 (2005), pp. 1440–1444.

[46] R. Penchovsky and R. Breaker. “Computational design and experimental validation of oligonucleotide-sensing allosteric ribozymes”. In: *Nat. Biotechnol.* 23.11 (2005), pp. 1424–1433.

[47] H. M. Salis, E. A. Mirsky, and C. A. Voigt. “Automated design of synthetic ribosome binding sites to control protein expression”. In: *Nat. Biotechnol.* 27.10 (2009), pp. 946–950.

[48] H. M. T. Choi, J. Y. Chang, L. A. Trinh, J. E. Padilla, S. E. Fraser, and N. A. Pierce. “Programmable in situ amplification for multiplexed imaging of mRNA expression”. In: *Nat. Biotechnol.* 28.11 (2010), pp. 1208–12.

[49] B. L. Li, A. D. Ellington, and X. Chen. “Rational, modular adaptation of enzyme-free DNA circuits to multiple detection methods”. In: *Nucleic Acids Res.* 39.16 (2011), e110.

[50] A. J. Genot, D. Y. Zhang, J. Bath, and A. J. Turberfield. “Remote Toehold: A Mechanism for Flexible Control of DNA Hybridization Kinetics”. In: *J. Am. Chem. Soc.* 133.7 (2011), pp. 2177–2182.

[51] A. J. Genot, J. Bath, and A. J. Turberfield. “Reversible Logic Circuits Made of DNA”. In: *J. Am. Chem. Soc.* 133.50 (2011), pp. 20080–20083.

[52] J. Choi, K. R. Love, Y. Gong, T. M. Gierahn, and J. C. Love. “Immuno-Hybridization Chain Reaction for Enhancing Detection of Individual Cytokine-Secreting Human Peripheral Mononuclear Cells”. In: *Anal. Chem.* 83.17 (2011), pp. 6890–6895.

[53] J. Dong, X. Cui, Y. Deng, and Z. Tang. “Amplified Detection of Nucleic Acid by G-Quadruplex Based Hybridization Chain Reaction”. In: *Biosens. Bioelectron.* 38.1 (2012), pp. 258–263.

[54] T. Nishimura, Y. Ogura, and J. Tanida. “Fluorescence resonance energy transfer-based molecular logic circuit using a DNA scaffold”. In: *Appl. Phys. Lett.* 101 (2012), p. 233703.

[55] M. Schade, A. Knoll, A. Vogel, O. Seitz, J. Liebscher, D. Huster, A. Herrmann, and A. Arbuzova. “Remote Control of Lipophilic Nucleic Acids Domain Partitioning by DNA Hybridization and Enzymatic Cleavage”. In: *J. Am. Chem. Soc.* 134.50 (2012), pp. 20490–20497.

[56] J. R. Vieregg, H. M. Nelson, B. M. Stoltz, and N. A. Pierce. “Selective nucleic acid capture with shielded covalent probes”. In: *J. Am. Chem. Soc.* 135.26 (2013), pp. 9691–9699.

[57] A. J. Genot, J. Bath, and A. J. Turberfield. “Combinatorial Displacement of DNA Strands: Application to Matrix Multiplication and Weighted Sums”. In: *Angew. Chem., Int. Ed.* 52.4 (2013), pp. 1189–1192.

[58] G. D. Hamblin, A. A. Hariri, K. M. M. Carneiro, K. L. Lau, G. Cosa, and H. F. Sleiman. “Simple Design for DNA Nanotubes from a Minimal Set of Unmodified Strands: Rapid, Room-Temperature Assembly and Readily Tunable Structure”. In: *ACS Nano* 7.4 (2013), pp. 3022–3028.

[59] C. C. Santini, J. Bath, A. M. Tyrrell, and A. J. Turberfield. “A clocked finite state machine built from DNA”. In: *Chem. Commun.* 49.3 (2013), pp. 237–239.

[60] H. M. T. Choi, V. A. Beck, and N. A. Pierce. “Next-generation in situ hybridization chain reaction: higher gain, lower cost, greater durability”. In: *ACS Nano* 8.5 (2014), pp. 4284–4294.

[61] Y. S. Jiang, S. Bhadra, B. L. Li, and A. D. Ellington. “Mismatches Improve the Performance of Strand-Displacement Nucleic Acid Circuits”. In: *Angew. Chem., Int. Ed.* 53.7 (2014), pp. 1845–1848.

[62] C. Geary, P. W. K. Rothemund, and E. S. Andersen. “A single-stranded architecture for cotranscriptional folding of RNA nanostructures”. In: *Science* 345.6198 (2014), pp. 799–804.

[63] A. A. Green, P. A. Silver, J. J. Collins, and P. Yin. “Toehold Switches: De-Novo-Designed Regulators of Gene Expression”. In: *Cell* 159.4 (2014), pp. 925–939.

[64] J. M. Hu, Y. J. Yu, J. C. Brooks, L. A. Godwin, S. Somasundaram, F. Torabinejad, J. Kim, C. Shannon, and C. J. Easley. “A Reusable Electrochemical Proximity Assay for Highly Selective, Real-Time Protein Quantitation in Biological Matrices”. In: *J. Am. Chem. Soc.* 136.23 (2014), pp. 8467–8474.

[65] R. R. F. Machinek, T. E. Ouldridge, N. E. C. Haley, J. Bath, and A. J. Turberfield. “Programmable energy landscapes for kinetic control of DNA strand displacement”. In: *Nat Commun* 5 (2014), p. 5324.

[66] E. Franco, G. Giordano, P. O. Forsberg, and R. M. Murray. “Negative Autoregulation Matches Production and Demand in Synthetic Transcriptional Networks”. In: *ACS Synth. Biol.* 3.8 (2014), pp. 589–599.

[67] B. Koos, G. Cane, K. Grannas, L. Lof, L. Arngarden, J. Heldin, C. M. Clausson, A. Klaesson, M. K. Hirvonen, F. M. S. de Oliveira, V. O. Talibov, N. T. Pham, M. Auer, U. H. Danielson, J. Haybaeck, M. Kamali-Moghaddam, and O. Soderberg. “Proximity-dependent initiation of hybridization chain reaction”. In: *Nat Commun* 6 (2015), p. 7294.

[68] J. G. Zalatan, M. E. Lee, R. Almeida, L. A. Gilbert, E. H. Whitehead, M. La Russa, J. C. Tsai, J. S. Weissman, J. E. Dueber, L. S. Qi, and W. A. Lim. “Engineering Complex Synthetic Transcriptional Programs with CRISPR RNA Scaffolds”. In: *Cell* 160.1-2 (2015), pp. 339–350.

[69] R. P. Galimidi, J. S. Klein, M. S. Politzer, S. Y. Bai, M. S. Seaman, M. C. Nussenzweig, A. P. West, and P. J. Bjorkman. “Intra-Spike Crosslinking Overcomes Antibody Evasion by HIV-1”. In: *Cell* 160.3 (2015), pp. 433–446.

[70] T. Raschle, C. X. Lin, R. Jungmann, W. M. Shih, and G. Wagner. “Controlled Co-reconstitution of Multiple Membrane Proteins in Lipid Bilayer Nanodiscs Using DNA as a Scaffold”. In: *ACS Chem. Biol.* 10.11 (2015), pp. 2448–2454.

[71] M. K. Takahashi, K. E. Watters, P. M. Gasper, T. R. Abbott, P. D. Carlson, A. A. Chen, and J. B. Lucks. “Using in-cell SHAPE-Seq and simulations to probe structure-function design principles of RNA transcriptional regulators”. In: *RNA* 22.6 (2016), pp. 920–933.

[72] Y. J. Lee, A. Hoynes-O’Connor, M. C. Leong, and T. S. Moon. “Programmable control of bacterial gene expression with the combined CRISPR and antisense RNA system”. In: *Nucleic Acids Res.* 44.5 (2016), pp. 2462–2473.

Chapter 4

Next-Generation Sequence Design for Nucleic Acid Reaction Pathway Engineering: Enhanced Models, Flexibility, and Speed

Molecular programmers are faced with the challenge of translating a conceptual program into a set of molecules to interface with the real world. Based on our work in Chapter 3, optimization of a set of target test tubes formulated to capture important reactant, intermediate, product and crosstalk states through the multistate test tube ensemble defect has been an effective way to obtain nucleic acid sequences implementing these molecular programs. We have shown that finding sequences that are both of high quality and highly constrained is achievable. Empirically, designs with stringent constraint sets require more computational time and achieve lower design quality relative to less constrained designs.

We introduce an enhanced algorithm that solves multistate sequence design problems more effectively. The design algorithm inherits the algorithmic and implementation improvements of the underlying thermodynamic calculations from Chapter 2 to give a baseline performance boost and more accurate modeling of coaxial and dangle stacking states. We introduce a soft constraint framework for multiobjective design to provide users with additional flexibility through new constraint types and lowered design cost when replacing hard constraints with their soft constraint variants in highly constrained designs. We introduce a new meta constraint satisfaction problem (CSP) solver using restart-based search to increase constraint solving speed in most cases and to robustly find feasible sequences in highly constrained cases. The hard constraint solving framework is augmented with a new diversity constraint, allowing easier and clearer specification of a common constraint set as well as better solver performance. These enhancements frequently lead to order of magnitude reductions in design cost relative to the algorithm and implementation of Chapter 3, and even greater reductions when switching to soft constraints in the most constrained designs.

4.1 Introduction

The design algorithm presented in Chapter 3 showcases the ability to design the thermodynamics of several local equilibrium states simultaneously all while using sequences that meet user-specified constraints, with design priorities modulated through user-defined defect weights. This is an important contribution because real world reaction pathway design problems often require all of these components, motivating their inclusion in the multistate test tube design algorithm. This formulation and functionality have been instrumental in designing new reaction pathway mechanisms[1–5] in test tubes, cell lystate, bacterial cells, and mammalian cells.

However, adding constraints had two general effects on the quality and cost of design. First, adding a single set of constraints to the design specification caused a deterioration in speed by up to two orders of magnitude. Second, adding all of the individual constraint sets simultaneously led to such stringent design conditions that design cost increased by up to four orders of magnitude and the target stop condition for $\mathcal{M}$, $f_{\text{stop}} = 0.02$, was no longer achieved. In practice these two effects are not necessarily rate limiting due to downstream experimental validation being more costly. Furthermore, principled design is still preferable to heuristic design followed by experimental filtering. So, we sought new methods to improve the computational performance of multistate test tube design.

From a new methodological standpoint, we implement soft constraints to complement the existing hard constraints of the multistate test tube design algorithm. One subset of these soft constraints is analogous to the pattern prevention and similarity hard constraints. For highly constrained designs, replacing hard constraints with soft constraints results in significantly lower design cost. The other subset of soft constraints allows for additional flexibility in design specification. These soft constraints introduce new functionality to design by allowing sequence symmetry minimization and equalization of the free energy of structures. These soft constraints fulfill common user requests for optimizing additional heuristic quantities for avoiding kinetic traps and balancing reactions rates of nucleic acid reaction pathways[6,7].

Another large boost to performance is inherited by switching to the new thermodynamics back-end implementation of Chapter 2. Such drastic increases in analysis speed shone light on other components of the design algorithm that were previously insignificant relative to the expensive thermodynamics calculations. This led to an investigation of the constraint satisfaction problem (CSP) solver algorithm we were using as well as development of a new meta solver to improve this performance bottleneck.

Finally, our reimplemented version of the multistate test tube design algorithm with the algorithm changes of this chapter reduced source code size by about 50%. This algorithm is packaged with a Python API, replacing the previous custom scripting interface and making the algorithm accessible within the wider Python scientific computing community as well as embeddable as a library inside other projects.

4.2 Formulation

4.2.1 Ensemble

Our physical design ensemble is defined the same way as in Section 3.2.2.

4.2.2 Multiobjective Optimization Problem Formulation

We define our objective function for multiobjective design as:

$$\mathcal{M} + \sum_{k \in \mathcal{S}} w_k f_k(\phi_\Psi). \tag{4.1}$$

Here, $\mathcal{M}$ is the multistate test tube ensemble defect defined in Section 3.2.4, and $\mathcal{S}$ is the set of soft constraints discussed in Section 4.2.4. The design problem formulated as a minimization problem then becomes

$$\min_{\phi_\Psi} \left[\mathcal{M} + \sum_{k \in \mathcal{S}} w_k f_k(\phi_\Psi) \right] \quad \text{subject to} \quad \mathcal{R}, \tag{4.2}$$

where $\mathcal{R}$ is the set of hard constraints discussed in Sections 3.2.5 and 4.2.3. As with the multistate test tube design algorithm, we do no seek a true minimizer of the objective function, but rather seek to find a *satisficing*[8] sequence ϕ^*_Ψ such that

$$\mathcal{M} + \sum_{k \in \mathcal{S}} w_k f_k(\phi^*_\Psi) \leq f_{\text{stop}}\,, \tag{4.3}$$

i.e. a sequence with a corresponding objective function value that "good enough" relative to the $f_{\text{stop}} \in (0, 1)$ threshold.

This type of optimization problem is refered to as a *multiobjective combinatorial optimization problem* in the optimization literature[9]. Our objective function is a *scalarized acceptance criterion*. This means that while we are ultimately trying to lower the $\mathcal{M}$ and the various $f_k(\phi_\Psi)$ values as much as possible, we select the sum of the individual defects as our proxy for "as much as possible." Other possible scalarized acceptance criteria include minimization based on the worst objective or based on some other norm (e.g. sum of squared defects)[10]. Scalarized acceptance criteria constrast with *component-wise acceptance criteria*, which compare solutions by looking at the individual objectives ($\mathcal{M}$ and the soft constraint $f_k(\phi_\Psi)$ values) and finding a set of solutions that is a mutually non-dominated approximation to the Pareto front[9]. Development effort started along the component-wise path; the output of such an algorithm is a set of solutions that have to be selected amongst by the user *a posteriori*. The generation of this set of solutions is also expected to take longer than producing a single satisficing sequence. Ultimately we changed to the above scalarized objective function and allow users to specify weights on the objectives (soft constraints) to bias the search toward a desired balance of the objectives *a priori*.

4.2.3 Hard Constraints

The algorithm continues to use a constraint satisfaction framework for sequence initialization and mutation. We continue to use the same set of sequence constraints described in Section 3.2.5. We now refer to the constraints defining the set $\mathcal{R}$ as *hard constraints* to distinguish from *soft constraints*, which are discussed in the next section. Specifically, hard constraints must be met exactly at every step during design, from sequence initialization through sequence mutation and reseeding. For comparison with the typical specification of an optimization problem, we write the objective function (Section 4.2.2) as a minimization over the sequence space ϕ_Ψ subject to $\mathcal{R}$. However, it is useful to think of $\mathcal{R}$ itself as uniquely defining the sequence space for the problem. When thought about this way, there is no sense of degree of violation of a constraint: a sequence is either in the search space or it is not. Furthermore, $\mathcal{R}$ and the CSP solver used to find valid sequences can be thought of as the sequence generator for the design algorithm.

In addition to the constraints of Section 3.2.5, we introduce the new *diversity constraint*, which allows for succint specification as well as efficient solution of a common constraint case, as discussed below.

Diversity Constraint. A diversity constraint ensures that for a given string of nucleotide variables (a domain or strand) every window of a given length has a certain mininum number of distinct nucleotide types. It is specified by the sequence, ϕ, which sets the scope of the constraint, and then two integer parameters: l_{window} and $n_{\min}$. The parameter l_{window} sets the size of the substring windows of ϕ to apply the constraint to. The parameter $n_{\min}$ sets the minimum number of distinct nucleotide types (`A`, `C`, `G`, `T`, or `U`) that must appear in each length l_{window} substring of ϕ.

The pattern prevention constraint, introduced in Section 3.2.5, could be used to specify a diversity constraint indirectly. For example, for the case studies of Section 3.4.1, the constraint set referred to as "pattern" in Figure B.19, was imposed by disallowing any four consecutive nucleotides from matching the patterns `AAAA`, `CCCC`, `GGGG`, `UUUU`, and any six consecutive nucleotides from matching the patterns `RRRRRR`, `YYYYYY`, `MMMMMM`, `KKKKKK`, `WWWWWW`, `SSSSSS`. The set of sequences meeting the above set of constraints is exactly equivalent to the set meeting two diversity constraints, the first with $l_{\text{window}} = 4$ and $n_{\min} = 2$ and the second with $l_{\text{window}} = 6$ and $n_{\min} = 3$. Specifying this set of sequences via diversity constraints instead of pattern constraints is simpler, the intent of the constraint is clearer, and, as shown in Section 4.4.4, the constraint solver can find satisfying sequences in less time.

4.2.4 Soft Constraints

We now introduce the concept of soft constraints. Unlike hard constraints which must be met exactly for every sequence considered during the design, soft constraints can be violated during design. For each soft constraint, one can also compute a *degree of violation*, which measures the extent to which a constraint has been violated. We also refer to this as the defect of the soft constraint by analogy to the multistate test tube ensemble defect, $\mathcal{M}$. This allows numerical comparison of the quality of sequences that violate the soft constraint. This constrasts with hard constraints where all violation leads to indistinguishably infeasible sequences. In this sense, soft constraints are auxiliary objective functions that it is the goal of our design algorithm to minimize alongside $\mathcal{M}$.

Generic Definition and Properties. We denote the set of soft constraints as $\mathcal{S}$. For each $k \in \mathcal{S}$, we define a function $F_k(\phi_\Psi)$ that evaluates the degree of violation for the constraint. This degree of violation is then normalized in a constraint-dependent fashion to be in $[0, 1]$. The function $f_k(\phi_\Psi)$ returns this normalized degree of violation. The normalized degree of violation is used during design instead of the raw degree of violation so that each soft constraint may be weighted relative to $\mathcal{M}$. These weights, w_k, fall in the range $[0, \infty)$ with a default of unity. Finally, for each nuleotide a in the scope of soft constraint k, we can calculate $f_k^a(\phi_\Psi)$, the contribution of this nucleotide to $f_k(\phi_\Psi)$. The values of $f_k^a(\phi_\Psi)$ are used during multiobjective defect weighted mutation sampling (Section 4.3.5). Many of these functions involve indicator functions, for which we use the Iverson bracket notation,

$$[S] \equiv \begin{cases} 0, & \text{if } S \text{ is false} \\ 1, & \text{if } S \text{ is true} \end{cases}, \tag{4.4}$$

where S is any mathematical statement.

We now discuss the individual soft constraint types. For each type, we describe the parameters necessary to define the constraint and show how $f_k(\phi_\Psi)$ and $f_k^a(\phi_\Psi)$ are calculated.

Soft Pattern Prevention Constraint. The soft pattern prevention constraint is specified using the same information as the hard pattern prevention constraint. Two pieces of information are needed to specify a soft pattern prevention constraint k: a pattern sequence r to prevent in a contiguous sequence of nucleotides ϕ, where $|\phi| = m$, $|r| = n$, and $m \leq n$. Together these implicitly specify a set of $n - m + 1$ subsequence windows, ϕ_k, each of length m. The value of the function $F_k(\phi_\Psi)$ is the number of these windows that match r. The number of windows, $n - m + 1$, is the normalization factor. Therefore, the normalized degree of violation is

$$f_k(\phi_\Psi) = \frac{1}{n-m+1} \sum_{w \in \phi_k} [w = r], \tag{4.5}$$

where $[w = r]$ is an indicator function that is one if the window w matches the pattern and the sum runs over all windows in ϕ_k.

This value of $f_k(\phi_\Psi)$ can be attributed to the individual windows in ϕ_k, with each window receiving a contribution of $\frac{[w=r]}{n-m+1}$. This contribution is then divided equally among the nucleotides in the window. This translates to

$$f_k^a(\phi_\Psi) = \frac{1}{m(n-m+1)} \sum_{w \in \phi_k} [a \in w][w = r] \tag{4.6}$$

for the contribution of nucleotide a to $f_k(\phi_\Psi)$.

Soft Similarity Constraint. Specifying the soft similiarity constraint requires the same information as the hard similiarity constraint. Specifically, it requires a chosen sequence of nucleotide variables, ϕ, a reference sequence of degenerate nucleotide codes of length $|\phi| = N$, q, and finally upper and lower limits on the number of nucleotides in ϕ matching q, u and l, respectively. The limits must be such that $0 \leq l \leq u \leq N$. The main quantity that is used for evaluation of the soft constraint is the generalized hamming distance,

$$d(\phi, q) = \sum_{a \in 1,\ldots,|\phi|} \begin{cases} 0 : \phi^a \in q^a \\ 1 : \phi^a \notin q^a \end{cases}. \tag{4.7}$$

We work instead with the number of nucleotides matching the reference sequence, $x = N - d(\phi, q)$. The degree of violation is simply the number of nucleotides additionally matching when $x > u$ or the deficiency in matching nucleotides when $x < l$. The normalization factor is defined as $c = \max(l, N - u)$. This ensures that the number of additional nucleotides matching or nucleotides not matching the reference sequence are treated equivalently. Additionally, in the case of either $x = N$ or $x = 0$, whichever involves more violations, $f_k(\phi_\Psi) = 1$. With this, the normalized degree of violation is

$$f_k(\phi_\Psi) = \frac{1}{c} F_k(\phi_\Psi)$$

$$F_k(\phi_\Psi) = \begin{cases} x - u, & x > u \\ l - x, & x < l \\ 0, & l \leq x \leq u \end{cases} . \tag{4.8}$$

When mapping down to the individual nucleotides, which nucleotides contribute defect depends on whether $x > u$ or $x < l$. When $x > u$, although only $x - u$ nucleotides are outside of the match range, every matching nucleotide is contributing to the defect because mutating any of them to a non-matching nucleotide would reduce $f_k(\phi_\Psi)$. Thus all of the matching nucleotides effectively share the constraint violation equally. Likewise for the non-matching nucleotides when $x < l$. This leads to the per nucleotide contribution to $f_k(\phi_\Psi)$ of

$$f_k^a(\phi_\Psi) = \frac{f_k(\phi_\Psi)}{x} \left[\phi^a \in q^a\right], \tag{4.9}$$

where q^a is the reference sequence position that matches the position of a in ϕ.

Sequence Symmetry. Sequence symmetry[11] was one of the first objective functions used for nucleic acid design, through sequence symmetry minimization (SSM)[12,13]. SSM is still used as a heuristic to avoid potential kinetic traps in the nucleic acid folding pathway[7]. More broadly, SSM is a heuristic method that implements a negative design paradigm by reducing the types of possible off-target structures that can form[14].

SSM was formulated for use in designing single nucleic acid complexes. The only defining parameters are the target sequence, s, and a word length, l_{window}. In this ensemble of the complex, perfect sequence symmetry is achieved by meeting 3 rules[11]:

1. Every subsequence window of length l_{window} must contain a unique sequence.
2. If w is a length l_{window} window, then its reverse complement window, w^*, can only appear if w and w^* are paired within a duplex in s.
3. Self-complementary sequences of length l_{window} cannot appear in any window.

Taken together, these ensure that any off-target binding results from either perfect matches of length less than l_{window} or involves interior loop or bulge loop structures, which are generally less favorable than perfect stack loops.

In the multi test tube ensemble, sequence symmetry continues to be a complex-level objective function. However, as the general design problem has multiple on-target complexes, the sequence symmetry soft constraint allows a scope of multiple complexes, a set $\Psi_k \subseteq \Psi^{\text{on}}$. These complexes in general have sequence domains that are constrained to have the same sequence or complementary sequences. As such, the above rules need to be generalized to this situation where some sequence repetition is unavoidable due to constraints while other repetition is spurious and should be penalized. In this expanded environment, where the set of all windows is W_k, the rules for perfect sequence symmetry become:

1. Every subset of W_k is constrained to be identical across all complexes $j \in \Psi_k$ must be the only subset of W_k with this sequence.
2. If any member of a subset c of W_k that is constrained to be identical across all complexes $j \in \Psi_k$ appears in a non-duplexed motif for some s_j, the reverse complement to the sequence of this window cannot appear anywhere except in the subset of W_k that is constrained to be complementary to the windows in c.
3. Self-complementary sequences of length l_{window} are not allowed, unless required to form by complementarity.

Ultimately, with the types of hard constraints allowed in $\mathcal{R}$, one would have to fully marginalize over all other variables to determine if two nucleotides are constrained to be either identical or complementary in all feasible sequences. That is, propagating explicit identity and complementarity constraints alone will not discover all implied identity and complementarity constraints because they interact with all other constraint types. This marginalization procedure is not computationally tractable, so instead, during evaluation of the constraint we consider the restriction of these rules to explicitly specified identity and complementarity constraints. The effect of this choice is to potentially set a non-zero lower bound on $f_k(\phi_\Psi)$ corresponding to windows that appear to be spuriously identical or complementary but are in fact constrained to be so.

For each nucleotide variable a in at least one of the windows in W_k, we maintain the lists I_a and C_a. These lists contain other variables that are constrained to be identical to or complementary to a, respectively. The target secondary structures s_j for $j \in \Psi_k$ are processed to find windows that are not wholly in a duplex motif. For these windows, their reverse complement must be prevented to minimize rule 2 violations. These windows are stored in RC.

Windows in W_k can be *aliased*: two distinct windows within a single complex or from separate complexes contain the same sequence of nucleotide variables (e.g. domain a appearing in complex j and in complex j'). In this sense, W_k is a multiset of the unique windows of sequence variables, and we define the set W'_k to contain this underlying set of sequence variable windows. Bookkeeping is done at the level of sequences (and their reverse complements) that appear in the windows of W_k. Specifically, for a sequence ϕ that appears in some element of W_k, w_ϕ is a set of windows from W_k containing all windows that have the sequence ϕ as well as windows in RC whose reverse complement is ϕ. The set w_ϕ can be partitioned such that each disjoint subset contains windows constrained to be identical to or complementary to every other member in the subset, determined via I_a and C_a. The function $c(w_\phi)$ counts the number of subsets generated by this partition. So, for $w_\phi = \{a, b, c, d, e\}$, if $a = d$, $b = c^*$, and $d = e$, then $c(w_\phi) = 2$. We can thus define the normalized degree of violation as:

$$f_k(\phi_\Psi) = \frac{1}{|W'_k|} \sum_{\phi \mid \substack{\exists w \in W_k \\ w = \phi}} c(w_\phi) - 1 + [\phi = \phi^*], \tag{4.10}$$

where $[\phi = \phi^*]$ is an indicator function checking if ϕ is its own reverse complement, adding a penalty if rule 3 is violated. The -1 is necessary because rule 1 is only violated if there is more than one disjoint subset in the partition of w_ϕ.

This can be mapped down to individual nucleotide variables as:

$$f_k^a(\phi_\Psi) = \frac{1}{|W'_k|} \sum_{\phi \mid \substack{\exists w \in W_k \\ w = \phi \\ a \in w}} \sum_{w \in w_\phi \mid a \in w} \sum_{i \in w \mid i = a} \frac{c(w_\phi) - 1 + [\phi = \phi^*]}{l_{\text{window}} |w_\phi|}. \tag{4.11}$$

This is ultimately the purpose of w_ϕ containing windows from W_k instead of W'_k. The largest of the disjoint subsets in w_ϕ corresponds to most appearances of the window in the physical ensemble and so should be penalized proportionally more. In this way, defect-weighted mutation sampling will preferentially mutate the underlying variables that appear most frequently.

Structure Free Energy Equalization. The preceeding soft constraints have all been defined purely at the sequence level, without using the thermodynamic model. This parallels the hard constraints which are also non-thermodynamic. We now introduce the structure free energy equalization soft constraint which does make use of the thermodynamic model. This soft constraint is defined by a set of sequences ϕ_k, a corresponding set of structures s_k, and a reference free energy ΔG_{ref}. The sequences and structures do not need to correspond to the complexes in Ψ^{on}. The reference free energy can be either a fixed user-defined constant when one or more complexes are in the soft constraint scope, or it can be the median free energy of the sequence-structure pairs when $|\phi_k| = |s_k| > 1$:

$$\Delta G_{\text{ref}} = \begin{cases} \Delta G_{\text{external}} \\ \underset{(\phi_i, s_i) \in \phi_k \oplus s_k}{\text{median}} (\Delta G(\phi_i, s_i)) \end{cases},$$

where $\oplus$ represents the direct sum of sets. In the latter case, the soft constraint biases the design toward sequences which minimized the variance of the set $\{\Delta G(\phi_i, s_i)\}$ A motivating use case for this soft constraint is to equalize the toehold binding energy across analogous complexes in several orthogonal reaction pathways as a proxy for the reaction rate of the initial unbinding step (typically rate-limiting),[6] with the goal of all orthogonal pathways performing with comparable rates[15]. This case highlights that ϕ_i need not be drawn from the complexes in Ψ^{on} as toehold sequences are typically only subsequences of complexes in Ψ^{on} and do not appear alone as on-targets.

Because free energies are unbounded real-valued quantities, naively measuring differences between $\Delta G(\phi_i, s_i)$ and ΔG_{ref} leads to an unbounded value for the degree of violation. Thus, we use a function which approaches unity as $|\Delta G(\phi_i, s_i) - \Delta G_{\text{ref}}| \to \infty$ and is identically zero when $|\Delta G(\phi_i, s_i) - \Delta G_{\text{ref}}| = 0$:

$$f_k^i(\phi_\Psi) = 1 - \exp\left(-\left|\frac{\Delta G(\phi_i, s_i) - \Delta G_{\text{ref}}}{\Delta G_{\text{scale}}}\right|\right),$$

where ΔG_{scale} is a global algorithm parameter that ensures the unweighted degrees of violation of all structure free energy equalization soft constraints are comparable and makes the total argument to the exponential function dimensionless. Empirically, we use $\Delta G_{\text{scale}} = 10$ kcal/mol. From this we can define our normalized degree of violation,

$$f_k(\phi_\Psi) = \frac{1}{|\phi_k|} \sum_i f_k^i(\phi_\Psi), \tag{4.12}$$

where the normalization constant is the number of sequences/structures, $|\phi_k| = |s_k|$.

When mapping down to the individual nucleotides in the sequences in ϕ_k, the contribution from each pair (ϕ_i, s_i) is divided uniformly amongst the nucleotides in ϕ_i. Mathematically, this is expessed as

$$f_k^a(\phi_\Psi) = \frac{1}{|\phi_k|} \sum_i \sum_{m \in \phi_i | m = a} \frac{1}{|\phi_i|} f_k^i(\phi_\Psi), \tag{4.13}$$

with the sums necessary because the same nucleotide could appear in several sequences and multiple times per sequence. Note that this will lead defect-weighted mutation sampling to preferentially target the sequences in ϕ_k that contribute the most to the degree of violation, but will not preferentially bias mutations toward particular nucleotides in a given ϕ_i.

A Note on Soft Constraints and Estimates. One of the satisfying features of the soft constraints introduced in this chapter is that they are not approximated during design to decrease design cost (as must be done with $\mathcal{M}$). In fact, it is not even clear how one would estimate these degrees of violation in an analogous way to hierarchical ensemble decomposition. The cost of evaluating these constraints is at most $O(N)$, which will be dwarfed by the $O(N^3)$ cost to evaluate estimates $\tilde{\mathcal{M}}_d$. The immediate implication of this lack of approximation is that once the $f_k(\phi_\Psi)$ values are evaluated at the leaves, they will not change during the remainder of steps leading to the evaluation of $\mathcal{M}$. This simplifies the changes necessary for the remainder of the algorithm: as soft constraints do not change with refocusing or redecomposition, they are not necessary for these processes, which continue on exactly as in the algorithm of Chapter 3.

4.3 Algorithm

The algorithm described in Section 3.3.1 and the algorithm of this chapter share much of their basic framework. The subroutine call graph implied by Algorithm B.1 is left unchanged. However, the logic of each subroutine is modified to greater or lesser extents. As such, we describe here changes relative to Section 3.3.1 and note when components of that algorithm are reused without modification.

4.3.1 Thermodynamics Backend

This algorithm piggybacks off of the thermodynamics algorithms discussed in Chapter 2. This contrasts with the algorithm of Chapter 3, which was developed and published prior to the work of Chapter 2. It benefits from the enhanced speed of our vectorized implementation (Figure 2.10). The additional speedups due to block caching and reuse across complexes (Figure 2.11) are irrelevant during non-root computation as the blocks within interior nodes are not generally repeated across complexes due to independent decompositions. Finally, the extended model including coaxial and dangle stacking (Section 2.2.3) is accessible, allowing for design over a more physically accurate state space. This last point is of particular importance due to the frequency of structures with nicked helix motifs, which are unduly penalized in the secondary structure model excluding coaxial and dangle stacking[16–22].

4.3.2 Constraint Satisfaction Problem Solver

The CSP solving algorithms for sequence initialization, mutation, and reseeding described in Section B.1.11 were built in-house[23] using only the C++ standard library. The original software author notes that this implementation was not optimized with additional data structures and caching for fast performance[24]. When using the old thermodynamics backend in conjunction with this in-house CSP solver, the thermodynamics subroutines were slow enough that overhead from solving the CSP was not noticeable in the benchmark case of complex design of structures from an engineered test set. However, when used in conjunction with early versions of this algorithm, which used the new thermodynamics backend, CSP solving overhead was noticeable in overall design performance, motivating investigation into alternative solvers.

Gecode Constraint Solver. There exist a number of general purpose constraint solver libraries compatible with the C++ language. Preliminary investigation indicated that the Gecode library[25] could serve the needs of our algorithm for a CSP solver with better speed. During assignment, the library stores the path of commits necessary to get from a partial solution with fewer assigned variables to one with more assigned variables, i.e. from closer to the root to closer to the leaves in the search tree. Then during backtracking, new partial solutions are generated by applying the commits along the path shared between the current and next node, only doing constraint propagation once at the final step. The algorithm also bisects long paths with an explicit caching of the state in the middle of the path, making repeated backtracking less costly.

Additionally, when propagating constraints, the Gecode propagation scheduler is able to determine the cost of propagating each constraint and schedule the constraints with a priority queue. This minimizes the number of times the most expensive constraints are propagated if variable domains change multiple times during propagation of less expensive constraints. Most of the constraint types in Section 3.2.5 can be efficiently implemented by chaining together simpler constraint types in Gecode. For the pattern prevention, window, and library constraints, efficient custom propagators were developed and added to the framework using Gecode's object oriented propagator interface.

All of these features contribute to a generally faster implementation (as shown in Figure 4.6) and so we use Gecode as the primary constraint solver framework during design.

Hybrid Heuristic Constraint Solver. Solving a CSP is typically done through a search process involving assignment of a valid value from the domain of an unassigned variable followed by propagation of the implications for this selection on the remaining unassigned variables[26]. This entails using heuristics for variable selection (i.e. which variable to assign next) and value selection (i.e. for this variable which value should be assigned). Additionally, it is common when doing search to use several heuristics in a process called *restart-based search* (RBS),[25] where search is terminated by a cutoff condition if a solution is not found using the first heuristic, and then the next heuristic is tried, etc. We make use of all of these components for our implementation of a constraint solving framework.

In our previous in-house solver, the heuristic used for value selection required a tentative assignment step followed by constraint propagation for each value in the current variable's domain. The number of additional required mutations as a result of each of these tentative changes is used to rank the changes and preferentially explore sequences with fewer additional necessary mutations. This sort of heuristic method conflicts with the optimizations in Gecode that make it efficient. Specifically, whereas the tentative assignments and propagations can be cached in the in-house solver, this cannot be done in Gecode, thus requiring recomputation of the expensive propagation step. Empirically, however, there are highly constrained cases where the heuristics in Gecode that are more efficient in the average case lead to longer search times than the in-house implementation.

Thus, rather than using multiple heuristics within the Gecode RBS framework, we build a custom RBS framework melding both Gecode and the in-house solver. First, the time limit for Gecode to run per CSP solve operation, t_{max}, is initialized, by default to 1 second. During each call to the CSP solver (for initialization or mutation), the Gecode solver is used to search for a solution until one is found or until $t_{\text{search}} > t_{\text{max}}$, in which case the Gecode solver is stopped. If the Gecode solver is stopped due to timeout, the in-house solver is then used to find a solution. A running average of the times per in-house CSP solve is stored during design as t_{avg}. After each use of the in-house solver, t_{max} is updated to $0.1t_{\text{avg}}$. In this way, anytime the Gecode solver finishes faster than t_{max}, we have achieved at least a $10\times$ speedup on CSP solving relative to the in-house solver alone, and anytime Gecode is stopped due to timeout, we only incur a 10% penalty relative to using the in-house solver alone. This leads to a massive upside for easy CSPs while bounding the downside for difficult CSPs. In the following sections we describe the heuristics each solver uses for variable and value selection.

Sequence Initialization. For Gecode variable selection, we select the next unassigned variable. For Gecode value selection, we select a value uniformly at random from the domain of the variable. For in-house variable selection, we select the next variable uniformly at random from the unassigned variables. For in-house value selection, we select a value uniformly at random from the domain of the variable.

Sequence Mutation. During sequence mutation, the current sequence to mutate from, ϕ, is used in the heuristics. For Gecode variable selection, we select the next unassigned variable. For Gecode value selection, for selected variable b, we choose the value b has in ϕ if this value is still in the domain of b, otherwise we select a value uniformly at random from the domain of b. For in-house variable selection, we select the next variable randomly with probability proportional to its weight, which changes dynamically during search in the manner described in Section B.1.11. For in-house value selection, we select the value which causes the fewest additional variables to no longer be able to match their value in ϕ, determined using trial assignment and constraint propagation.

Sequence Reseeding. As described in Section B.1.11, sequence reseeding is simply iterated sequence mutation with a predefined set of positions to mutate. Therefore, for each position, the RBS method of sequence mutation is used.

4.3.3 Hierarchical Ensemble Decomposition

Here we detail the hierarchical ensemble decomposition algorithm[27] of Chapter 3 in anticipation of changes for the algorithm of this chapter.

The partition function for each complex $j \in \Psi$ can be calculated in $O(|\phi_j|^3)$. However, as the algorithm makes mutations in the strand sequences in an attempt to minimize $\mathcal{M}$, a naive implementation would require recalculation of the partition functions of all complexes which contained the strand(s) for which the sequence changed. Instead, the algorithm uses an effective estimation method called hierarchical ensemble decomposition[27].

Hierarchical ensemble decomposition recursively splits a complex ensemble into pairs of independent child ensembles with an enforced base pair flanking the split-point position in each child ensemble. The first level of decomposition can be seen graphically in Figure 4.1. Because pseudoknotted structures have been excluded from consideration, if the base pairs flanking the split-point form with high probability, then the ensembles of the children would be nearly disjoint Thus, their individual conditional partition functions are good predictors of the full parent partition function, but can be evaluated at lower cost. In cases for which no single split-point has high probability flanking base pairs, a set of exclusive split-points (read: mutually crossing) with a corresponding set of flanking base pairs whose summed probabilities approach unity can be used for the same effect. The process is recursive in the sense that each child containing an eligible split-point can be further decomposed using the same method.

We discuss the multiple split-point case for generality[27]. We use F to denote a single split-point and $\{F\}$ to denote a set of exclusive split-points. A conditional partition function contribution for each node k and each split-point $F_i \in \{F\}$ is calculated given that the base pairs flanking the split-point form with probability unity, resulting in a nodal partition function contribution $\tilde{Q}_{k_i}$. For split-point F_i with child nodes k_{l_i} and k_{r_i}, the estimate for this conditional ensemble's contribution to the

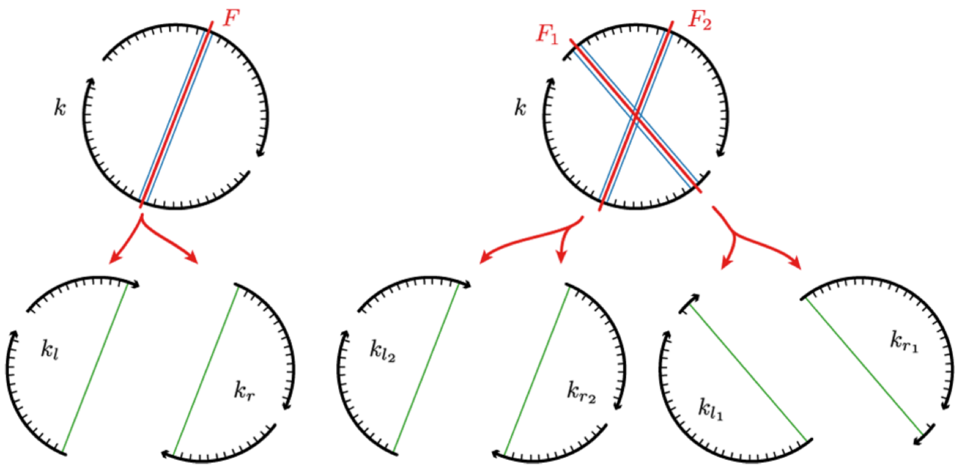


Figure 4.1: Polymer graph representation of ensemble decomposition with *left*: a single split-point and *right*: two exclusive split-points. The red lines are the split-points, the blue lines are base pairs forming with high probability, and the green lines are the same base pairs from the parent ensemble which are forced to form in the children[27]. Image from Reference 27.

estimate $\tilde{Q}_k$ can be calculated as

$$\tilde{Q}_{k_i} = \tilde{Q}_{k_{l_i}} \tilde{Q}_{k_{r_i}} \exp(\Delta G^{\text{interior}}_{F_i}/kT), \tag{4.14}$$

where $\Delta G^{\text{interior}}_{F}$ is the free energy of the interior loop through which the split-point divides the complex. The total partition function estimate for k is just the sum of the contributions for each mutually exclusive subensemble:

$$\tilde{Q}_k = \sum_{F_i \in \{F\}} \tilde{Q}_{k_i}. \tag{4.15}$$

From these nodal conditional partition functions, one can obtain an estimate for the complex partition function, $\tilde{Q}_j$ by applying the formula recursively until the root node is reached.

Additionally, the pair probability estimates for each pair of child nodes, $\tilde{P}_{k_{l_i}}$ and $\tilde{P}_{k_{r_i}}$, are combined into an estimate of the pair probabilities in the ensemble where the two base pairs flanking F_i are constrained to form. This is done by simply translating the entries from $\tilde{P}_{k_{l_i}}$ and $\tilde{P}_{k_{r_i}}$ to $\tilde{P}_{k_i}$ where they match the indexing nucleotides in node k. The contributions from the exclusive ensembles implied by $\{F\}$ are then combined by Boltzmann weighting in the following way:

$$\tilde{P}_k = \sum_{F_i \in \{F\}} \tilde{P}_{k_i} \frac{\tilde{Q}_{k_i}}{\tilde{Q}_k} \tag{4.16}$$

The estimate $\tilde{P}_k$ is then recursively propagated up to the root level estimate, $\tilde{P}_j$.

Finally, for each tube $h \in \Omega$ the $\tilde{Q}_j$ for each $j \in \Psi_h$ combined with the implied strand concentrations from the specification of $y_{h,j}$ for $j \in \Psi^{\text{on}}_h$ can be used to calculate estimates of the concentrations, $\tilde{x}_{h,j}$.

We now turn to the three possible scenarios for hierarchical ensemble decomposition[27].

Structure-Guided Decomposition with a Single Split-Point. Before sequence initialization begins, none of the complexes in Ψ^{active} have any pair probability information. However, at this point $\Psi^{\text{active}} = \Psi^{\text{on}}$, and each complex $j \in \Psi^{\text{on}}$ has a secondary structure, s_j. Thus, only this target structure information will be used to decompose the complex.

Given a target structure matrix S_k for some node to be decomposed, the set $B(S_k)$ contains all valid split-points F, the conditions for which are shown in (4.17). In this equation, $F^{\pm}$, represents the set of H_{split} base pairs on either side of the split-point. The top condition means that any potential split-point must have all base pairs in $F^{\pm}$ in S_k. The symbols ϕ_{k_l} and ϕ_{k_r} indicate the sequences of the parent node k that are divided into the left and right child nodes, respectively. The second condition requires that both child nodes contain at least N_{split} nucleotides. Since target structures do not have any pseudoknots, there is no possibility of multiple exclusive split-points meeting the requirements in (4.17), hence only the single split-point set.

$$B(S_k) \equiv \left\{ F\colon \begin{array}{c} \min\limits_{a \cdot b \in F^{\pm}} S_k^{a,b} = 1 \\ \min(|\phi_{k_l}|, |\phi_{k_r}|) \geq N_{\text{split}} \end{array} \right\} \tag{4.17}$$

Of these split-points consistent with the structure, we are interested in the lowest cost split-point,

$$F^* \equiv \arg\min_{F \in B(S_k)} (|\phi_{k_l}|^3 + |\phi_{k_r}|^3).$$

This procedure can be applied recursively on each new child node. Each new child node receives the truncated version of its parent's structure matrix that corresponds to its fragment of the complex. The recursive splitting continues until $B(S_k) = \varnothing$ for some node k, at which point k is a leaf node and decomposition stops for that branch of the tree.

Probability-Guided Decomposition with Multiple Exclusive Split-Points. Off-targets are first decomposed if they have been added to Ψ^{active} following ensemble refocusing (Section B.1.9). Because off-targets have no target structure, only probability information is used to decompose the ensemble. Additionally, because we are using the pair probability matrix of a node, P_k, we will generally need more than one exclusive split-point to capture $f_{\text{split}} Q_k$. Thus, we are interested in sets of split-points $\{F\}$ falling in the set $\overline{B}(P_k)$, described in (4.18). The top condition says that the *collective probability* captured by a valid set must exceed f_{split}. The middle condition is the same as in (4.17), regarding child nodes being large enough. The bottom condition requires that all split-points in a given set be mutually exclusive.

$$\overline{B}(P_k) \equiv \left\{ \{F\}\colon \begin{array}{c} f_{\text{split}} \leq \sum\limits_{F_i \in \{F\}} \min\limits_{a \cdot b \in F_i^{\pm}} P_k^{a,b} \\ \min\limits_{F_i \in \{F\}} (|\phi_{k_{l_i}}|, |\phi_{k_{r_i}}|) \geq N_{\text{split}} \\ F_i \otimes F_j \ \forall F_i \neq F_j \in \{F\} \end{array} \right\} \tag{4.18}$$

Of these sets of exclusive split-points, we are interested in the lowest cost set,

$$\{F\}^* \equiv \underset{\{F\}\in\overline{B}(P_k)}{\arg\min} \sum_{F_i\in\{F\}} (|\phi_{k_{l_i}}|^3 + |\phi_{k_{r_i}}|^3).$$

While generally $|\{F\}^*| > 1$, it is possible that a single split-point will meet the collective probability requirement.

This procedure can be applied recursively on each new child node. The pair probabilities are recomputed in the child ensembles to generate $P_{k_{l_i}}$ and $P_{k_{r_i}}$ conditioned upon the enforced base pairs flanking the split-points. The recursive splitting continues until $\overline{B}(P_k) = \varnothing$ for some node k, at which point k is a leaf node and decomposition stops for that branch of the tree.

Structure- and Probability-Guided Decomposition with Multiple Exclusive Split-Points. During redecomposition of on-target complexes (Section B.1.8), both pair probability and structure information are available to use for decomposition.

$$\hat{B}(S_k, P_k) \equiv \left\{\{F\}\colon \begin{array}{r} \{F\} = G_i \cup \{G\}_j, G_i \in B(S_k), \{G\}_j \in \overline{B}(P_k) \\ F_i \otimes F_j \ \forall F_i \neq F_j \in \{F\} \end{array}\right\} \tag{4.19}$$

Of these sets of exclusive split-points with one split-point consistent with the structure, we are interested in the lowest cost set,

$$\{F\}^* \equiv \underset{\{F\}\in\hat{B}(S_k,P_k)}{\arg\min} \sum_{F_i\in\{F\}} (|\phi_{k_{l_i}}|^3 + |\phi_{k_{r_i}}|^3).$$

This procedure can be applied recursively on each new child node. The new child node receives the truncated version of its parent's structure matrix that corresponds to its fragment of the complex. The pair probabilities are recomputed in the child ensembles to generate $P_{k_{l_i}}$ and $P_{k_{r_i}}$ conditioned upon the enforced base pairs flanking the split-points. In the nodes generated from the probability-guided component, further decomposition is done using probability-guided decomposition only. The recursive splitting continues until $\hat{B}(S_k, P_k) = \varnothing$ for some node k, at which point k is a leaf node and decomposition stops for that branch of the tree.

Branch and Bound Algorithm. In the above three cases, a branch and bound procedure is used to find the minimum cost split-point, F^*, or set of split-points, $\{F\}^*$. In the structure-guided, single split-point case, this algorithm reduces to linear search through all of the valid split-points in the fully enumerated $B(S_k)$ to find the minimum cost split-point.

In the probability-guided, multiple split-point case, the procedure starts by determining all split-point positions that are at least H_{split} away from the first and last index and resulting in children with more than N_{split} nucleotides. For each F, the quantity

$$\min_{a\cdot b\in F^{\pm}} P_k^{a,b}$$

is calculated to represent the fraction of the probability this split captures. All of these potential splits are ranked in descending order of this value. The trial set $\{F\}$ is initialized to $\varnothing$. From here, $\{F\}$ is expanded by adding the next highest probability split-point that is consistent with those already in $\{F\}$ This is done until

$$f_{\text{split}} \leq \sum_{F_i \in \{F\}} \min_{a \cdot b \in F_i^{\pm}} P_k^{a,b},$$

at which point $\{F\} \in \overline{B}(P_k)$. The cost of this first valid set of split-points then becomes the upper bound for pruning branches in the remainder of the search tree. The search proceeds in this way: popping the last added split-point when no other valid and unpruned split-points are consistent with the current set and attempting to add new consistent split-points when they do not cause the cost of $\{F\}$ to exceed the current best cost. Anytime during the search that a new split-point F_i would cause the cost of $\{F\}$ to exceed the cost of the current best, it is skipped and the next mutually-exclusive split-point is considered. This is valid because adding split-points only increases the cost, so if that split-point would cause the cost to exceed the current best, so would any superset. Anytime during the search where $\{F\} \in \overline{B}(P_k)$, if the cost of this valid set is lower than the current best cost, the best cost is changed to this value. At the end, the minimum cost $\{F\}^*$ will be found while many potentially valid but high-cost

The structure- and probability-guided, multiple split-point case is similar to the probability-guided case. However, the first split-point added to the trial $\{F\}$ is taken from $B(S_k)$. Anytime during the search that $\{F\} = \varnothing$, the next value from $B(S_k)$ is considered. The behavior when $|\{F\}| > 1$ is exactly as described for the probability-guided case. The search ends when $\{F\} = \varnothing$ and there are no more splits in $B(S_k)$ to consider.

Use in Design Initialization. At the start of design, no thermodynamic quantities have been computed. Thus, only structure-guided, single split-point ensemble decomposition is available. Applying this method recursively to all $j \in \Psi^{\text{active}}$, one is left with a decomposition forest, where each tree corresponds to one on-target complex. Each node in each tree has a depth d, and the deepest depth of any node across all forest is D. We call the set of all nodes Λ and the set of nodes at depth d (including these copied nodes) Λ_d. Since these trees can in general be unbalanced and the deepest node in each tree will be a different d, leaf nodes in Λ_d are copied as children recursively until all paths to a leaf node are length D. This makes discussion of computing at a given depth d always have the property that conditional quantities can be computed by looking solely at nodes in Λ_d.

As it is expected that the decomposition will introduce false negatives (i.e. high probability structures that are excluded in the child ensembles), we build in a tolerance for these missed structures by setting a per-depth stop condition f_d^{stop} according to the formula

$$f_d^{\text{stop}} \equiv f_{\text{stop}}(f_{\text{stringent}})^{d-1} \quad \forall d \in \{1, \ldots, D\},$$

where $f^{\text{stringent}} \in (0, 1)$. The intent is that over-designing the leaf nodes relative to f_{stop} will offset emerged defects that occur during merging (Section B.1.8) and full ensemble evaluation (Section B.1.9).

Asymptotic Optimality Lower Bound for Design. In the best case for complex design, the hierarchical ensemble decomposition above results in a balanced binary decomposition tree (i.e. no multiple split points and all nodes at any given depth have the same sequence length). In the optimal case, the initial sequence meets the leaf stop condition (B.11). Furthermore, during subsequence merging, each parent stop condition (B.13) is met, allowing the sequences to merge from the leaves to the root without any redecomposition and reoptimization. This means that the minimum cost of design is bounded below by the cost of evaluating the thermodynamic properties at each node. The cost to evaluate the partition function, pair probabilities, and ensemble defect has complexity $\Theta(N^3)$[28]. Therefore, if the root has a sequence ϕ of length N, these properties cost cN^3 to evaluate, where c is a scale factor. For this minimal cost case, each child has half the sequence length of its parent. The cost of evaluating every node in the tree of depth D is then[28]:

$$\begin{aligned}\sum_{d=1}^{D} 2^{d-1} c \left(\frac{N}{2^{d-1}}\right)^3 &= \sum_{d'=0}^{D-1} 2^{d'} c \left(\frac{N}{2^{d'}}\right)^3 \\ &= cN^3 \sum_{d'}^{D-1} \frac{1}{4^{d'}} \\ &< cN^3 \sum_{d'}^{\infty} \frac{1}{4^{d'}} \\ &= \frac{4}{3} cN^3.\end{aligned} \tag{4.20}$$

Or, in other words, the minimal cost for complex design is $4/3$ the cost of evaluating the thermodynamic properties of the root node. This is a striking result given the perceived difficulty of design relative to analysis in most engineering disciplines. Even more amazing is that this bound is approached in practice when using our previous design algorithm implementations[23,27,28] to design complexes of increasing size. Therefore, we sought for the algorithm of this chapter to continue to produce this pleasing result.

4.3.4 Modifications to the Hierarchical Ensemble Decomposition Procedure

The preceding section detailed the components of the hierarchical decomposition used in Chapter 3 and Reference 27. This chapter's algorithm makes several changes to components of the decomposition subroutine of Section 4.3.3 for simplicity and efficiency.

Split-Points Coincident with Base Pairs. The first change relative to Section 4.3.3 is in the definition of a split-point and the resultant change in computing partition function estimates. In the previous discussion, split-points are conceived as dividing a node such that the sequences of both child nodes were disjoint. This leads to two child ensembles each with their own base pair that is enforced to form with probability 1. Additionally, in computing the contribution of one split-point's child nodes to the parent node partition function using (4.14), the free energy of the implied stack loop between the enforced base pairs is needed. We aimed to make the description and the algorithm simpler at the cost of a small increase to computational cost during evaluation.

The key requirement for child ensembles is that they are independent. This is clearly the case when their sequences are independent. However, as discussed in the description of the backtrack-free base-pairing probabilities matrix algorithm in Section 2.3.10, an unpseudoknotted secondary structure ensemble is partitioned into disjoint subensembles by a base pair $i \cdot j$ with probability 1. The sequences of both the left and right ensembles both include the two nucleotides i and j. This repeated sequence does not lead to overlapping structures, however, because the loops that can form on either side of the base pair are disjoint.

Leveraging the above discussion, the algorithm now uses split-points which are coincident with base pairs. An example single-level, single split-point decomposition of this type is shown in Figure 4.2. As this eliminates the implied stack loop between the two base pairs flanking the split-point in the previous definition, (4.14) is simplified to

$$\tilde{Q}_{k_i} = \tilde{Q}_{k_{l_i}} \tilde{Q}_{k_{r_i}}, \tag{4.21}$$

which is the inverse of (2.16) where $P_{i,j} = 1$. In assigning the base pairs in $\tilde{P}_{k_{l_i}}$ and $\tilde{P}_{k_{r_i}}$ to $\tilde{P}_{k_i}$, the only change is that the value of 1 for the split-point base pair $a \cdot b$ will be transfered twice, once from each child matrix. In the code, this is implemented as plain assignment, so the second 1 overwrites the first 1 and the value remains correct. The relations in (4.15) and (4.16) remain the same.

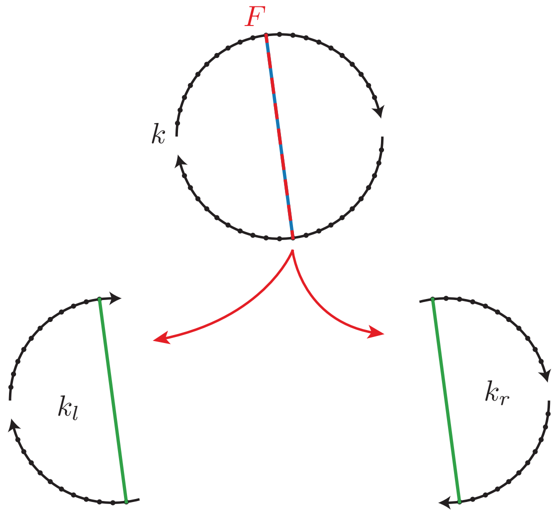


Figure 4.2: Polymer graph representation of ensemble decomposition with a split point along a base pair. The split point, which is also the base pair being split along, is shown as a red and blue dashed line. The green lines are the same base pair from the parent ensemble which are forced to form in the children.

As mentioned, these changes to the algorithm come at a slight computational cost increase. If a node k has N nucleotides and is split using the split-point generation from Section 4.3.3 so that one child has l nucleotides, then the total cost of computing the two child ensembles would be $(N - l)^3 + l^3$. With this section's algorithm, if one of the child nodes has length $N - l$, the other has length $l + 2$ for a cost of $(N - l)^3 + (l + 2)^3$. The difference in costs is $O(N^2)$. In Section 4.4.3, we show that this difference is only noticeable for small sequences and negligible for larger sequences.

Cost Bounding for Probability-Guided and Structure- and Probability-Guided Decomposition. With the branch and bound procedures discussed in Section 4.3.3 for probability-guided and structure- and probability-guided hierarchical ensemble decomposition, it is possible to generate a set of child nodes that is more expensive to compute than its parent. In fact, any decomposition with 4 split-points is guaranteed to be at least as expensive as evaluating the parent, and a decomposition with 5 or more split-points will always be more expensive. To see this, note that the least costly split points divide the N nucleotides into two children with $N/2$ nucleotides each. The cost of one of these pairs is then

$$2\left(\frac{N}{2}\right)^3 = \frac{N^3}{4}.$$

Thus, 4 minimum cost split-points will yield a collective evaluation cost of N^3 for children, equivalent to the parent.

As noted in the description of the branch and bound algorithm, the procedure is guaranteed to find a non-empty set of split-points that meets the collective probability requirement if such a set exists. The cost of this first discovered decomposition then is used to exclude exploration of more costly solutions. However, since this first decomposition was not itself bounded, its cost could exceed the parent cost. The implementation of the branch and bound procedure[23], imposed a limit on the depth of the search tree to 6 pairs of nodes, but, as discussed above, this is too permissive.

Instead of bounding the depth of the branch and bound search tree directly, we use the parent node k itself (the case of no further decomposition) as the first explored solution. This solution has cost N^3, which is then used as the minimum cost bound for all further solutions. This naturally bounds the depth of the search tree and ensures that only decompositions that are less expensive than the parent (and meet the collective probability requirement) are considered. Otherwise, decomposition terminates on the branch ending in k as a leaf node.

Given the discussion in the previous section about the increased cost of child ensemble calculation due to the base pairs repeated in both children, how does this interact with the more aggressive bounding used here? When splitting along a base pair, the minimum cost to evaluate both children resulting from a single split-point exceeds $\frac{N^3}{4}$, so the maximum number of pairs of children is 3. However, in the analysis for split-points without repeated pairs, any non-minimal cost split-points lowers the maximum number of split points to at most 3. Thus, in typical cases both algorithms have the same effective maximum depth bound of 3.

Sparse Pair Probabilities Matrix. One component of the algorithm of Chapter 3 whose mechanics were not discussed in depth is how the conditional pair probabilities matrices are stored and manipulated. In the previous multistate test tube design algorithm implementation[23], the thermodynamics backend[29] call to compute pair probabilities returns the dense upper triangle of the symmetric pair probabilities matrix. The design algorithm implementation then filters this sparse matrix into a coordinate list sparse matrix format. We now examine why, for efficiency purposes, it is important to use sparse matrices during design.

For a complex j with N nucleotides and a decomposition tree Λ_j, we can define the quantity

$$E_{\Lambda_j} = \sum_{a=1}^{N} \sum_{b=1}^{N} \left[\tilde{P}_j^{a,b} > 0 \right],$$

which is the number of non-zero elements in the root estimate $\tilde{P}_j$. For a sparse matrix, we need store only these E_{Λ_j} elements. In our implementation, we further reduce the entries stored in the sparse matrix by thresholding out probabilities that are too low, by default less than 10^{-5}. We will continue to use the number of non-zero elements as an upper bound of the cost. For the ideal case of a balanced binary decomposition where the leaf nodes all have N_{split} nucleotides, each leaf can contribute at most N_{split}^2 non-zero elements to $\tilde{P}_j$. This leads to an upper bound on the smallest E_{Λ_j} of

$$\begin{aligned} \min_{\Lambda_j}(E_{\Lambda_j}) &\leq (N_{\text{split}})^2 \frac{N}{N_{\text{split}}} \\ &\leq N N_{\text{split}}. \end{aligned}$$

Because each leaf node entry is copied once per level and Λ_j has depth $\log_2(N/N_{\text{split}})$, the total cost for transfering the entries from leaf to root is $\log_2(N/N_{\text{split}}) N N_{\text{split}}$ The cost of computing the leaf node partition functions is

$$(N_{\text{split}})^3 \frac{N}{N_{\text{split}}} = N N_{\text{split}}^2.$$

Therefore, the asymptotic ratio of the time to copy the matrices to the time to compute the partition functions is

$$\frac{\log_2(N/N_{\text{split}}) N N_{\text{split}}}{N N_{\text{split}}^2} = \frac{\log_2(N/N_{\text{split}})}{N_{\text{split}}},$$

or $O(\log(N))$, since N_{split} is constant.

This is in contrast to using dense matrices for the same decomposition, where the time necessary to copy from leaf to root is equal to the storage of all intermediate matrices (since they must all be copied one time into their parent matrix). This has an asymptotic limit of

$$\begin{aligned}\sum_{i=1}^{\infty} 2^i \left(\frac{N}{2^i}\right)^2 &= N^2 \sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^i \\ &= \frac{N^2}{2} \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i \\ &= \frac{N^2}{2}(2) \\ &= N^2.\end{aligned}$$

As the cost of computing the leaf node partition functions is the same, the asymptotic ratio of the time to copy the matrices to the time to compute the partition functions is

$$\frac{N^2}{N N_{\text{split}}^2} = \frac{N}{N_{\text{split}}^2},$$

or $O(N)$, since N_{split} is constant. Empirically, in earlier prototype versions of the implementation, dense matrices were used for pair probabilities merging and the effects of the asymptotic scaling were visible in design runtime results.

Our implementation makes explicit use of the Armadillo matrix library[30] to handle sparse matrix operations through its underlying compressed sparse column storage format. Using a third party library alleviates the need to program a matrix framework by hand, which is an enteprise fraught with the potential for introducing bugs. Additionally, the library provides idiomatic and efficient ways of performing transforms on sparse matrices, including expansion to a dense matrix when needed. This leaves the appearance of our implementation of the mathematical operations closer to the math itself.

4.3.5 Leaf Optimization

The initial leaf sequence is found through the methods of Section 4.3.2. Evaluation of the leaf level estimate, $\tilde{\mathcal{M}}_D$, follows the treatment of Section B.1.7. The logic based on the parameters M_{bad}, M_{reopt}, and M_{reseed} has the same structure, except sequences are compared based on the modified objective function estimate described in the next section. Sequence mutation and sequence reseeding use the methods of Section 4.3.2 along with an updated defect-weighted mutation sampling procedure described two sections hence.

Soft Constraint Evaluation. After each mutation, the defects of the soft constraints in $\mathcal{S}$ are calculated using (4.5), (4.8), (4.10), and (4.12). Combined with $\tilde{\mathcal{M}}_D$ this allows us to calculate the leaf level estimate of the objective function:

$$\tilde{\mathcal{M}}_D + \sum_{k \in \mathcal{S}} w_k f_k(\phi_\Psi). \tag{4.22}$$

This the allows us to define the updated leaf level stop condition as:

$$\tilde{\mathcal{M}}_D + \sum_{k\in\mathcal{S}} w_k f_k(\phi_\Psi) \le f_D^{\text{stop}}. \tag{4.23}$$

It is important to note that all of the soft constraints are evaluated exactly at the leaves. Therefore, when more precise estimates include the term $\sum_{k\in\mathcal{S}} w_k f_k(\phi_\Psi)$, it should be understood that these values have been cached at the leaf level and are simply referenced in the other estimates instead of being recomputed.

Multiobjective Defect-Weighted Mutation Sampling. Mutations continue to be targeted preferentially toward nucleotides contributing the most to the objective function. However, we must now consider the defects of the soft constraints in addition to $\tilde{\mathcal{M}}_D$. Instead of sampling a nucleotide a with probability (B.12), we use (4.6), (4.9), (4.11), and (4.13) to define the following probability:

$$\frac{\tilde{\mathcal{M}}^a + \sum_{k\in\mathcal{S}} w_k f_k^a(\phi_\Psi)}{\tilde{\mathcal{M}}_D + \sum_{k\in\mathcal{S}} w_k f_k(\phi_\Psi)}, \tag{4.24}$$

where

$$\tilde{\mathcal{M}}^a = \sum_{h\in\Omega} \sum_{j\in\Psi_h^{\text{on}}} \tilde{\mathcal{M}}_{h,j}^a$$

marginalizes over all on-target complexes containing nucleotide a in all target test tubes in which these on-target complexes appear. This captures both soft constraint and $\tilde{\mathcal{M}}_D$ contributions.

4.3.6 Merging and Redecomposing

During subsequence merging of sequences at level $d+1$ to level d, Equation(B.13 now incorporates soft constraints and becomes:

$$\tilde{\mathcal{M}}_d + \sum_{k\in\mathcal{S}} w_k f_k(\phi_\Psi) \le \max(f_d^{\text{stop}}, \frac{1}{f_{\text{stringent}}}(\tilde{\mathcal{M}}_{d+1} + \sum_{k\in\mathcal{S}} w_k f_k(\phi_\Psi))). \tag{4.25}$$

When failing to meet this condition at any level d, redecomposition ensues, followed by additional leaf optimization.

Redecomposition Based on $\tilde{\mathcal{M}}_d$. As the values of $f_k(\phi_\Psi), k \in \mathcal{S}$ do not change as one merges up the decomposition forest, we use precisely the same redecomposition procedure as described in Section B.1.8.

4.3.7 Full Ensemble Evaluation

Similar to the changes made for subsequence merging, in evaluating the full objective function and checking the assumptions behind ensemble focusing our termination stop condition is modified from (B.14) to be:

$$\mathcal{M} + \sum_{k\in\mathcal{S}} w_k f_k(\phi_\Psi) \le \max(f_{\text{stop}}, \tilde{\mathcal{M}}_1 + \sum_{k\in\mathcal{S}} w_k f_k(\phi_\Psi)). \tag{4.26}$$

If this condition is not met, the ensemble must be refocused.

Refocusing Based on $\mathcal{M}$. As the values of $f_k(\phi_\Psi)$, $k \in \mathcal{S}$ do not change between the approximation by Ψ^{active} and the full ensemble, we use precisely the same refocusing procedure as described in Section B.1.9.

4.3.8 Defect Weights

Defect weights for $\mathcal{M}$ are handled in the same way as in Section B.1.6.

As mentioned in Section 4.2.4, each soft constraint is associated with a weight upon specification. Unlike $\mathcal{M}$, which is a hierarchical and global quantity for the design, each soft constraint has a user-specified scope. For this reason, there are no weights of the type used within $\mathcal{M}$ to prioritize within a soft constraint. Such a use case can be replicated by instead applying two separate soft constraints to disjoint subsets of the original soft constraint and giving each soft constraint a different weight. The intention is that increasing w_k will cause the design to prioritize nucleotides contributing to $f_k(\phi_\Psi)$, thereby leading to a lower final value.

4.4 Results and Discussion

4.4.1 Trials

All benchmarks were run using a single computational core on AWS EC2 C5 instances (3.0 GHz Intel Xeon Platinum processors) with 72 GB of memory for the complex design cases and 8 GB memory for the reaction pathway design test cases.

4.4.2 Model and Recursions Used

Many of the following results are generated using the same specifications as previous results shown in Chapter 3 or Appendix B. The "some" dangles heuristic used in NUPACK 3.2 design algorithm to produce the previous results was also used in the following results generated with the NUPACK 4.0 design algorithm. This was done to allow a fair comparison in terms of the structural ensemble used. Thus, although the design algorithm of this chapter inherits the ability to compute over the full secondary structure model including dangle and coaxial stacking states, the results that follow do not showcase this capability.

4.4.3 Hierarchical Ensemble Decomposition Performance Comparison

We sought to examine the effects of our changes to the hierarchical ensemble decomposition algorithm, discussed in Section 4.3.4. To do this, we used two test sets. The first test set is the engineered dimer test set consisting of structures of two equal length strands totalling 50, 100, 200, and 400 nt. For each size, there are 50 unique structures. The second test set is the engineered single-stranded test set consisting of structures of one strand of length 100, 200, 400, 800, 1600, and 3200 nt. For each size, there are 30 unique structures. Five design trials were run for each structure in both test sets using stop condition $f_{\text{stop}} = 0.01$, using RNA at 1 M Na$^+$ at 37 °C. For each trial, the ensemble consists of a single test tube with only the heteroduplex or single strand present as an on-target and no off-target complexes ($|\Psi| = |\Psi^{\text{on}}| = 1$). For both of these test sets, structures were generated to have helix lengths, loop sizes, and numbers of branches per loop that are characteristic of engineered structures from the nucleic acid nanotechnology and dynamic programming literature[27].

During design with the new algorithm, the process emitted serialized representations of the decomposition tree of the on-target complex after initial structure-guided decomposition and after any structure- and probability-guided redecomposition. These serialized representations were then processed by a separate program to compute the relative cost of computing the entire decomposition tree to the cost of computing the root node:

$$\left(\sum_{k \in \Lambda_j} |\phi_k|^3\right) / |\phi_j|^3,$$

where $|\phi_j|$ is the number of nucleotides in complex j and $|\phi_k|$ is the number of nucleotides in a node k of the decomposition tree for j, Λ_j. During the design with the old algorithm, the process emitted only these relative costs after each decomposition or redecomposition event. This cost assumes perfect cubic scaling, which is approximately true over the relevant range of lengths (Figure 2.10). In reality, the cost gets proportionally larger for longer sequences because of switching to overflow-safe code (Sections 2.3.6 and 2.3.8). So this function is actually an upper bound on the relative cost. It is also useful in its own right as an implementation invariant way to compare two decompositions.

The results are shown in Figure 4.3. For consistency with Chapter 2, we refer to the algorithms by the version of NUPACK they are a part of, so 3.2 for the previous algorithm and 4.0 for the new algorithm. While there may be more than one decomposition during design, we consider only the costs to evaluate the initial and final decompositions. Also, in cases not requiring redecomposition, these will be the same. The initial cost allows us to comment purely on interactions between the structures and the algorithms as no thermodynamic or randomized information is involved. The final cost allows us to place a lower bound on the total cost of design, as we know every node in this tree was evaluated for design to exit. In examining the results, comparing the shapes of the initial decomposition distributions for the old and new methods shows little difference. As discussed in Section 4.3.4, the costs are slightly higher for the new method due to repeated nucleotides in sibling nodes resulting in a rightward stretching of the distributions. Also, as expected, this stretching becomes negligible as complex size increases.

Note that in the results for the 3200 nt complexes in the single stranded test set using the previous algorithm, only 7 data points are displayed. These represent the trials which finished in time to submit this document. They are shown here to corroborate the claim that some members of the set of 3200 nt target structures have a lower bound, even when using the previous decomposition algorithm. This is clear from the points in the initial decomposition panel which have costs exceeding $2\times$ the root.

Ultimately, it is difficult to reason deeply about how the recursive decomposition algorithm will interact with a given structure *a priori*. Thus, we used the above analysis to characterize the efficiency over test sets of representative engineered complexes. Additionally, we can get a sense of

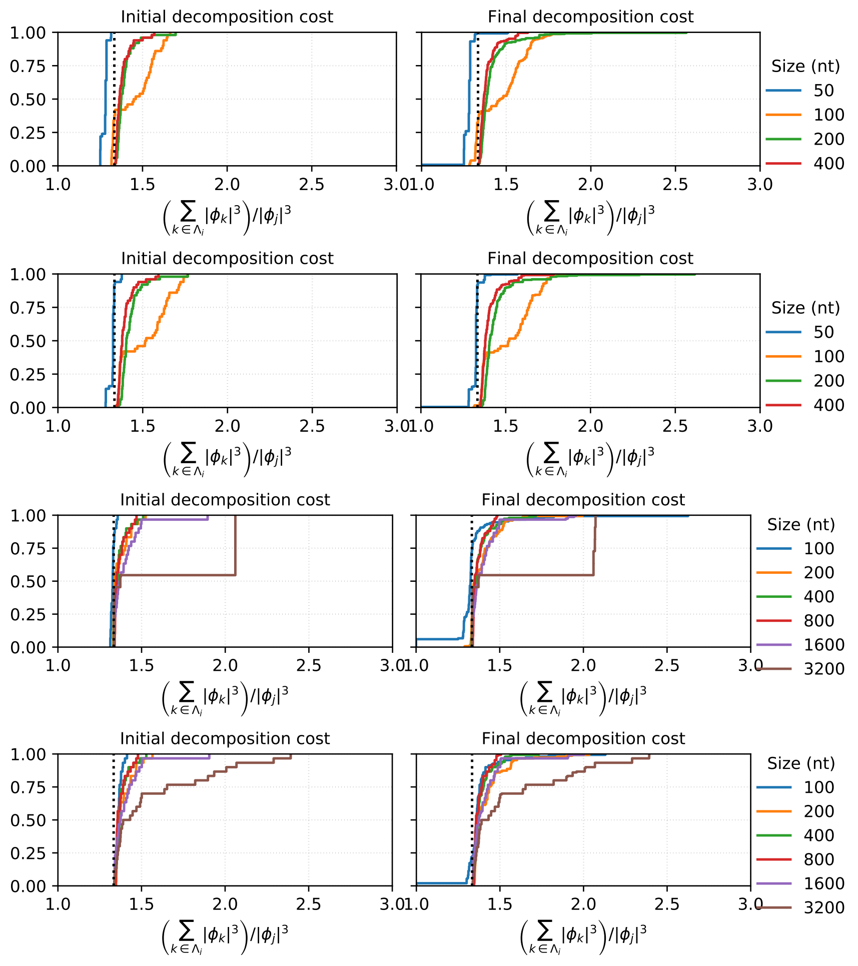


Figure 4.3: Comparison between NUPACK 3.2 and NUPACK 4.0 hierarchical decomposition algorithms on the engineered complex test sets. For each test set and algorithm pairing, two quantites are computed. Left panel: Idealized cost of computing every node in the initial structure-guided hierarchical ensemble decomposition relative to the root. Right panel: Idealized cost of computing every node in the final structure- and probability-guided hierarchical ensemble decomposition relative to the root. Each test set and algorithm pairing are shown on their own rows. First row: performance of the 3.2 algorithm on the engineered heterodimer complex test set. Second row: performance of the 4.0 algorithm on the engineered heterodimer complex test set. Third row: performance of the 3.2 algorithm on the engineered single-stranded complex test set. Fourth row: performance of the 4.0 algorithm on the engineered single-stranded complex test set.

the types of issues causing expensive decompositions using our decomposition visualization utility (Appendix C), by which we are able to see the decomposition trees for any complex. There is essentially one way that a decomposition can be efficient (i.e. approach the 4/3 lower bound (4.20)): if it is binary (no multiple split points) and balanced (each parent is divided directly in half). A visualization corresponding to the final decomposition tree of the most efficiently decomposed 3200 nt complex is shown in Figure 4.4. While the tree is not perfectly balanced, it is nearly so, especially closest to the root node, where this is most important.

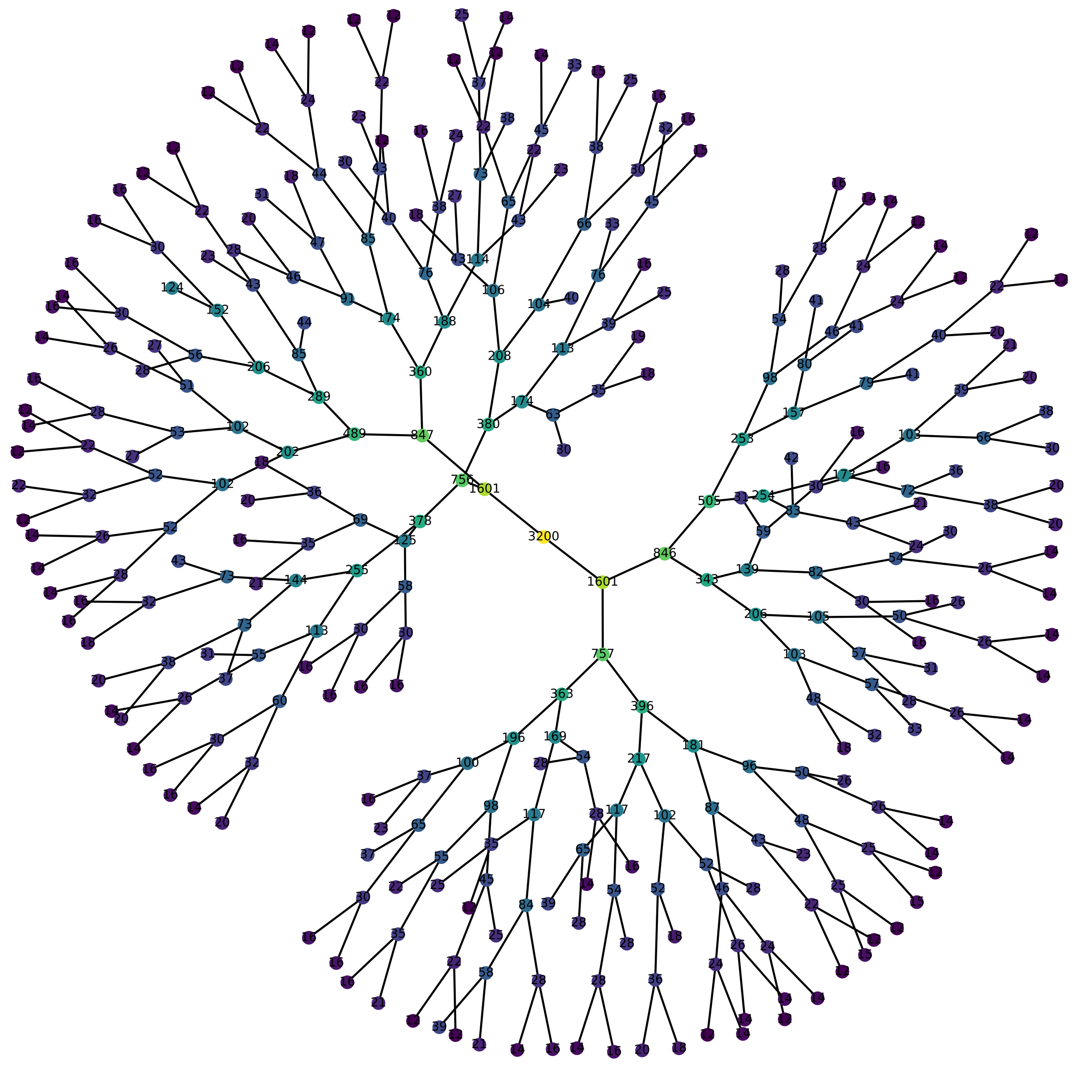


Figure 4.4: Example efficient decomposition of 3200 nt structure. Assuming perfect cubic scaling, evaluating the entire tree costs 1.34× the amount of time it takes to evaluate the root node. Nodes are labeled by their number of nucleotides, $|\phi_k|$. Nodes are colored with a perceptually uniform scale by $\log |\phi_k|$ to make use of the color scale's dynamic range.

The two ways a decomposition can be inefficient are failures of the requirements for efficiency: there are competing structural ensembles that must be captured with multiple exclusive split points or the least costly base pairs to decompose along do no neatly bisect the sequence into equal length subsequences leading to an unbalanced tree. From Figure 4.3 we can already see that the second type of inefficiency is predominant because the initial decomposition distributions (which do not use thermodynamic information) show inefficiency, which is especially pronounced for the 3200 nt structures. Figure 4.5 shows the final decomposition tree of the most inefficiently decomposed 3200 nt complex. This example decomposition was particularly egregious in that there is a central "trunk" in the tree with branches that peel off asymmetrically all the way down to the leaves. The features of this tree are representative of the other expensive decompositions, namely asymmetric splits at depth 1 or 2 in the tree. This was the typical failure mode observed in the most expensive final decompositions for every length. Inefficiency due to multiple split points in these test sets appeared negligible, with multiply-decomposed nodes typically appearing nearer to the leaves than the root.

It is likely that the particular causes of expensive decompositions will vary across test sets and target structures. However, users interested in causes of expensive decompositions for their particular design can introspect the causes by visualizing the trees in their design's decomposition forest.

4.4.4 Constraint Satisfaction Solver Performance

We now examine the effects of our change in CSP solving algorithm described in Section 4.3.2. Here, we examine raw CSP solving speed, without confounding thermodynamics calculations. Later, we examine the effects of CSP solving speed in the context of overall design cost for complex design and multistate test tube design.

To examine pure CSP solver mutation cost, for increasing domain lengths, we created a domain of that length and associated it with another domain constrained to be its reverse complement. We refer to these domains as a and a^*. Without any other constraints, this is the "complementarity" constraint set. For the "composition" constraint set, we add to the complementarity specification that domain a must have 45-55% GC content. For the "pattern" constraint set, we add to the complementarity specification that the patterns `AAAA`, `CCCC`, `GGGG`, `UUUU`, `RRRRRR`, `YYYYYY`, `MMMMMM`, `KKKKKK`, `WWWWWW`, and `SSSSSS` must not appear in domain a. For the "diversity" constraint set (only implemented in the hybrid solver), we impose the same effective constraint set as the pattern constraint, but using diversity constraints. For the "window" constraint set, we add to the complementarity specification that the first 50 nucleotides of a are constrained to be a subsequence of the 1798 nt *desma* mRNA sequence from *Danio rerio*[31]. Results for these different constraint sets and implementations are shown in Figure 4.6.

We see two major trends. First, the scaling of the in-house solver algorithm is asymptotically worse than the hybrid solver. Second, the absolute mutation times for the in-house solver are between 1 and 3 orders of magnitudes slower than their hybrid counterparts. The only exceptions to this are for the hybrid pattern and hybrid window constraints sets for shorter domain lengths. We expect that the diversity constraint would be used to implement the equivalent pattern constraint set, and so hybrid pattern performance is not a problem for smaller sequences in practice. The window constraint set

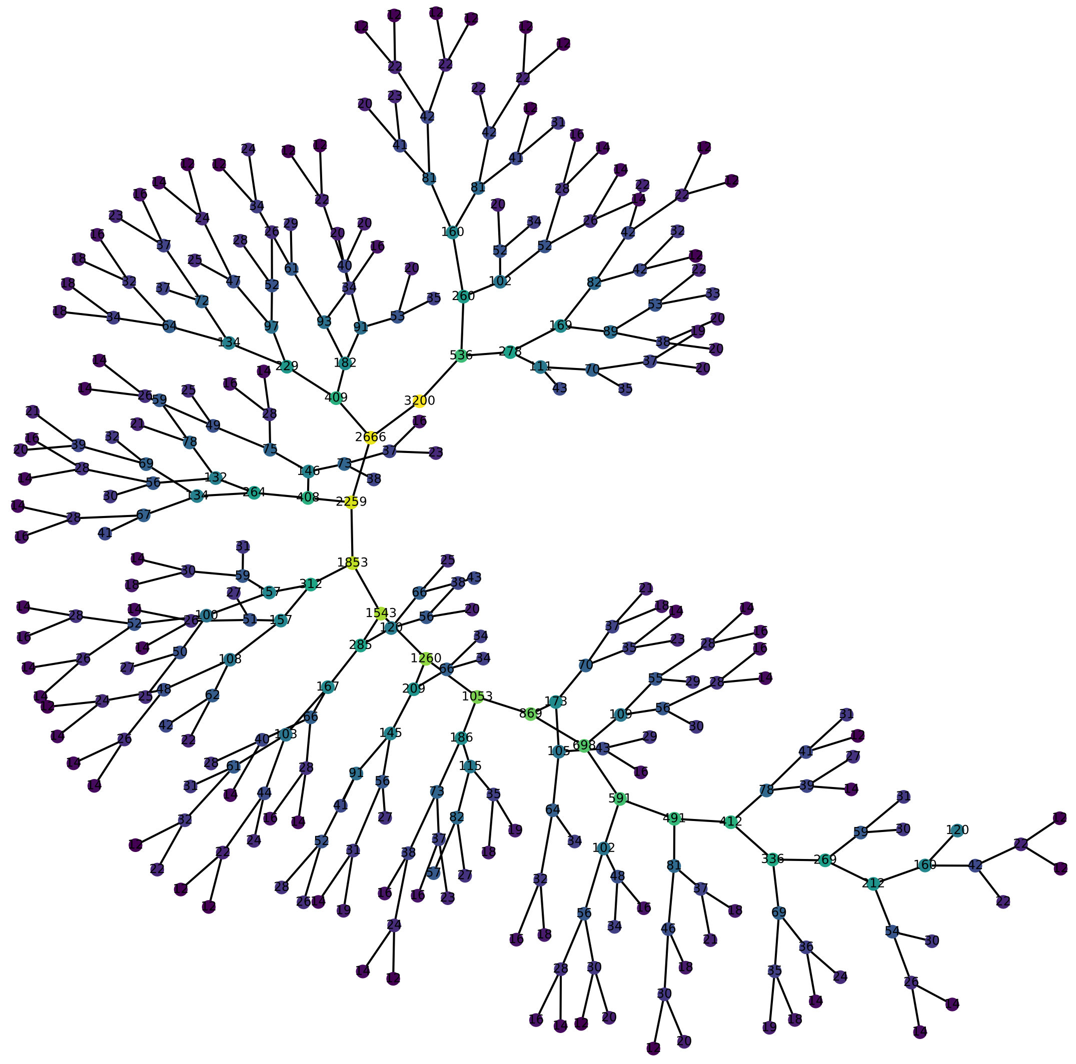
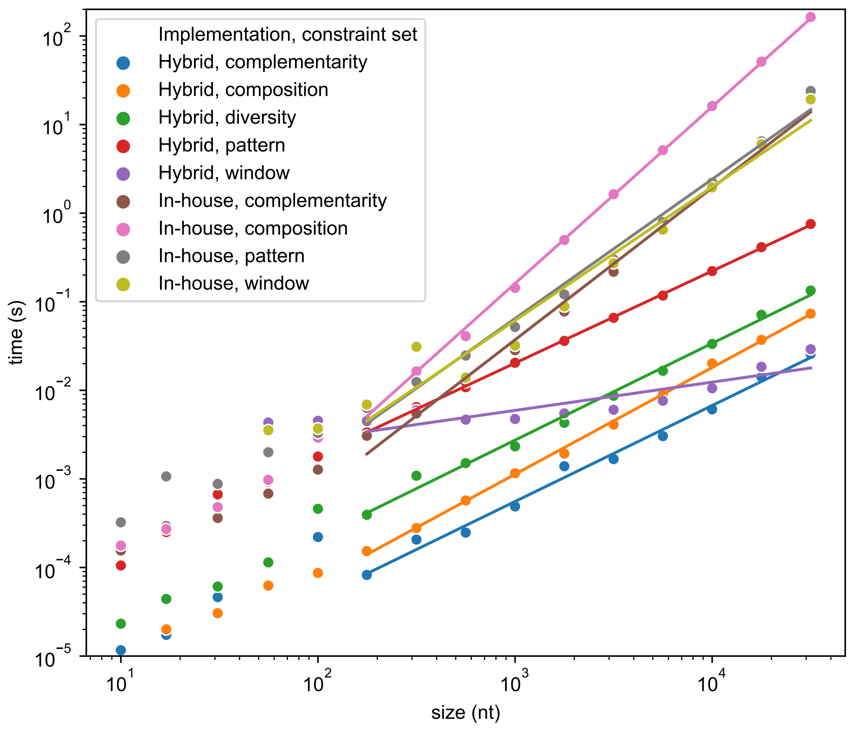


Figure 4.5: Example inefficient decomposition of 3200 nt structure. Assuming perfect cubic scaling, evaluating the entire tree costs 2.39× the amount of time it takes to evaluate the root node. Nodes are labeled by their number of nucleotides, $|\phi_k|$. Nodes are colored with a perceptually uniform scale by $\log|\phi_k|$ to make use of the color scale's dynamic range.



Implementation	Contraint set	Complexity in N	Prefactor (s)
Hybrid	complementarity	1.08	3.12×10^{7}
	composition	1.20	2.74×10^{7}
	diversity	1.09	1.44×10^{6}
	pattern	1.04	1.55×10^{5}
	window	0.32	6.54×10^{4}
In-house	complementarity	1.71	2.67×10^{7}
	composition	1.99	1.67×10^{7}
	pattern	1.57	1.23×10^{6}
	window	1.51	1.86×10^{6}

Figure 4.6: Scaling of CSP mutation cost with size of sequence (in nucleotides) for both in-house and hybrid CSP solvers. Dots represent mean times over 5 mutations, where the mutations for each point started from the same constraint-satisfying sequence. Lines represent power law fits for each constraint/implementation pairing for points with > 100 nt. Data for the “window” series only include sequences longer than 50 nt.

is particularly expensive for both solvers, but adds only a fixed cost, not changing the asymptotic scaling relative to the default.

4.4.5 Complex Design Performance

We now consider benchmark results for complex design of engineered target structures. In Section 4.4.3 we discussed running designs to investigate the quality of decompositions generated with the previous and new decompositions. We now examine the final outputs of those design trials, allowing performance comparison between the previous and new design algorithms.

The results are shown in Figure 4.7. First, the design quality for all trials was lower than $f_{\text{stop}} = 0.01$ stop condition. Additionally, for a given test set, the distributions of design quality for each size are visually similar between the two algorithms. In terms of raw performance, we consider the evaluation cost. We see speedups from 10-30× coming from the improvements of Chapter 2. This contributes to speedups of total design cost in the range of 2-25×. This differential speedup between design time and analysis time means when examining relative design cost, we see that the 4.0 algorithm has worse relative design costs, despite faster absolute times. This is due to overhead from the non-thermodynamic design code becoming apparent as the thermodynamic code has been sped up. However, the differences become inconsequential for complexes of 800 nt or larger, and indeed we see asymptotic approach to the 4/3 lower bound (4.20).

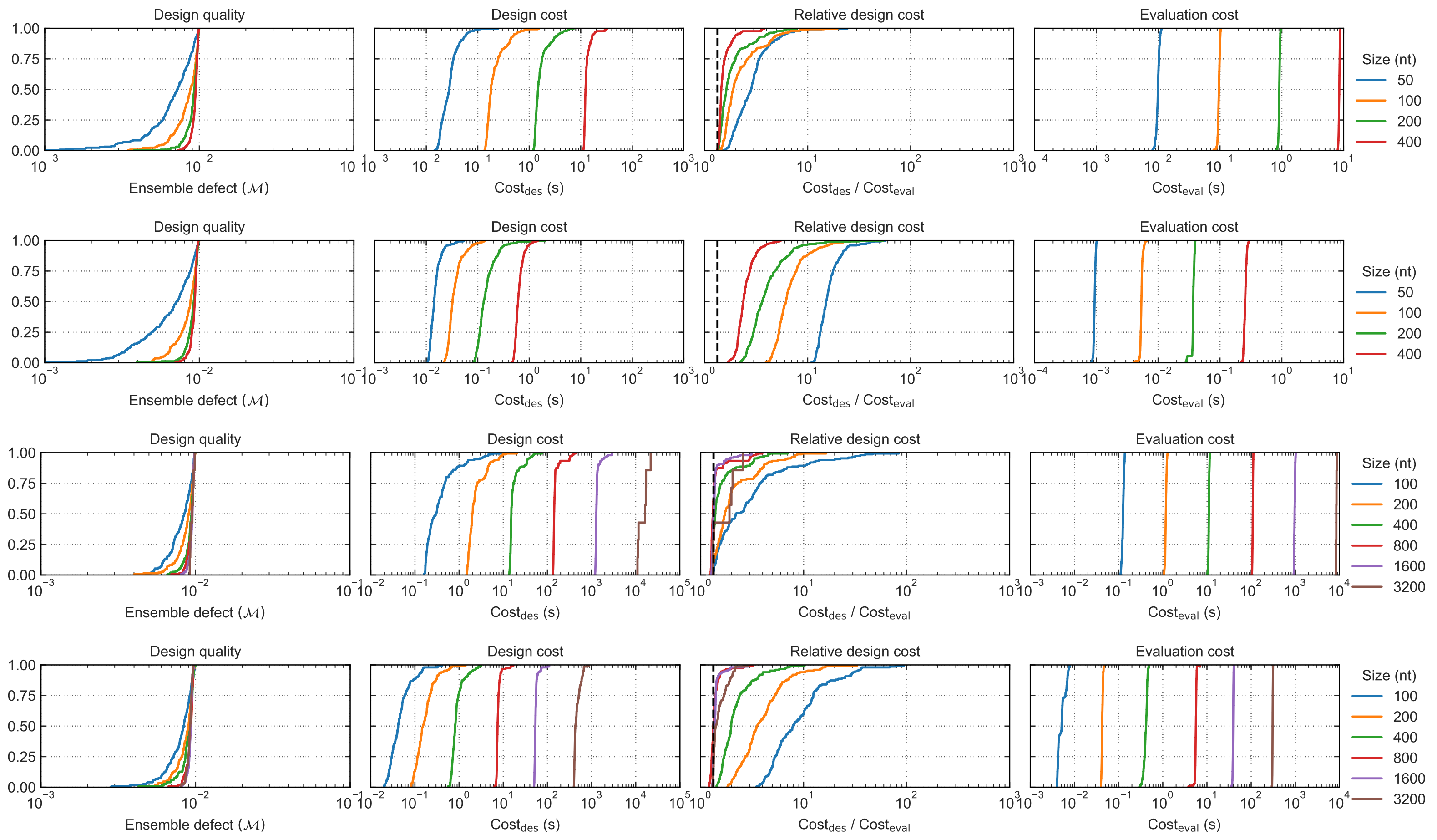


Figure 4.7: Performance of the NUPACK 3.2 and NUPACK 4.0 design algorithms on the engineered complex test sets. For each combination of algorithm and test set, four quantities were measured. First panel: Design quality. Second panel: Design cost. Third panel: Cost of sequence design relative to a single evaluation of the objective function. Fourth panel: Cost of a single evaluation of the objective function. Each test set and algorithm pairing are shown on their own rows. First row: Performance of the 3.2 algorithm on the engineered heterodimer complex test set. Second row: Performance of the 4.0 algorithm on the engineered heterodimer complex test set. Third row: Performance of the 3.2 algorithm on the engineered single-stranded complex test set. Fourth row: Performance of the 4.0 algorithm on the engineered single-stranded complex test set. Note that in the results for the 3200 nt complexes in the single stranded test set using the 3.2 algorithm, only 7 data points are displayed.

Effect of Constraint Performance On Design Cost for Complex Design. Now we consider the downstream effects of CSP solving on overall design performance. During development when using only the in-house constraint implementation, representative 3200 nt test cases were profiled to determine the fraction of time spent solving the CSP to initialize or mutate the sequence. For the first case, profiling a design with only structural constraints showed that 15% of the total design time was spent solving the CSP. For the second case, profiling a design with structural constraints and an imposed GC content of 45-55% showed that an astounding 63% of the total design time was spent solving the CSP. If we assume a median value of 30 for the speedup on the thermodynamic backend due to the improvements of Chapter 2 and consistent with Figure 4.7, then the equivalent fractions using the previous thermodynamics backend would have been 0.6% and 5.4%, respectively. The fraction of time spent solving the CSP during mutation was 91.8% and 99.2%, respectively, meaning that only at most 8.2% and 0.8% were spent on the thermodynamics calls during mutation. The equivalent fractions with the previous thermodynamics implementation would have been 27% and 81%, respectively. Thus, the improvements in performance for computing thermodynamic quantities were so successful that the fractional contribution of solving the CSP went from negligible to appreciable. Nevertheless, the fractions of time spent solving the CSP at the mutation level went from appreciable to dominant. These two points acted as strong motivation for finding and using a more efficient constraint solver.

We expanded the above analysis across the test sets of Section 4.4.3 and 4.4.5. While we produced data for the complete test set, we show here only the data for the 3200 nt test cases. From a decomposition perspective, these cases should be the most efficient in terms of relative design cost. Based on the results in Figure 4.6, however, CSP solving cost scales superlinearly with number of nucleotide variables for the in-house solver. We therefore sought to discover which of these effects dominated when using either the in-house or hybrid solver variants.

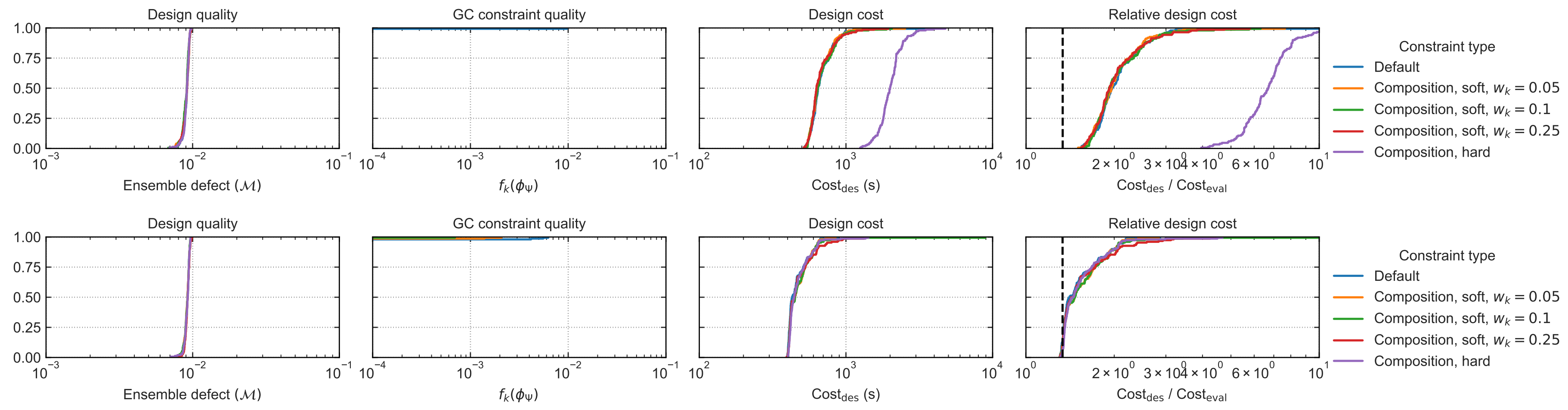


Figure 4.8: Comparison of design cost with the in-house CSP solver vs. the new hybrid CSP solver on 3200 nt test cases with differing levels of composition constraint. For each solver variant, distributions of four quantities were determined across the design trials. First panel: Design quality. Second panel: Quality with respect to the soft composition constraint (normalized degree of violation). Third panel: Design cost. Fourth panel: Cost of sequence design relative to a single evaluation of the objective function. Each row contains the results for one solver variant. Top: Results with the 4.0 algorithm using only the in-house CSP solver. Bottom: Results with the 4.0 algorithm using the hybrid CSP solver.

Figure 4.8 compares overall design performance using the in-house CSP to that using the hybrid CSP solver on cases involving composition constraints. The default series corresponds to only structure-based constraints, as in Figure 4.7. The other series correspond to constraining GC content to 45-55%, either through a hard constraint or a soft constraint with one of three different weights. Considering first the results using the in-house solver, we see that even the lowest relative design costs for the default series do not approach the 4/3 lower bound (4.20). Additionally, the distributions for the default series and all three soft constraint series are visually identical. However, there is a roughly 3-fold increase in design cost when using the hard composition constraints. This is unattractive because the default series is mostly meeting the requirement, as shown by the distributions in the second panels. Turning to the results with the hybrid solver, we first notice that absolute design cost has shifted lower for all series relative to those associated with the in-house solver. We also see asymptotic approach of the default series to the 4/3 lower bound. In fact, the distributions of design cost and relative design cost are mostly overlapping for all of the series. This better coincides with the aforementioned point that most trials in the default series meet the composition constraint without enforcement.

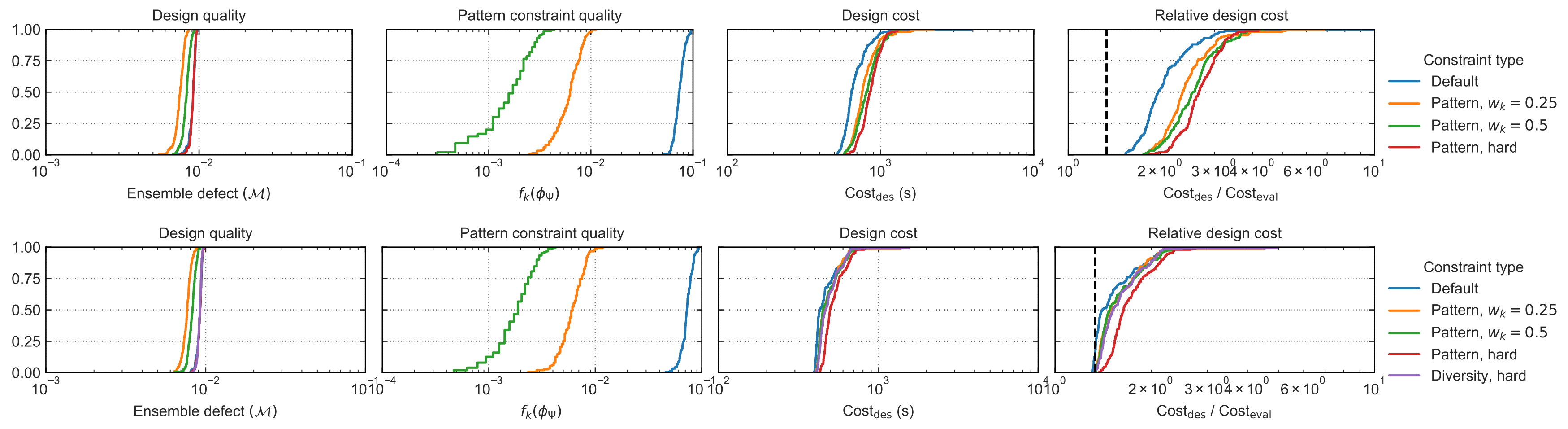


Figure 4.9: Comparison of design cost with the in-house CSP solver vs. the new hybrid CSP solver on 3200 nt test cases with differing levels of pattern prevention constraint. For each solver variant, distributions of four quantities were determined across the design trials. First panel: Design quality. Second panel: Quality with respect to the soft pattern constraint (normalized degree of violation). Third panel: Design cost. Fourth panel: Cost of sequence design relative to a single evaluation of the objective function. Each row contains the results for one solver variant. Top: Results with the 4.0 algorithm using only the in-house CSP solver. Bottom: Results with the 4.0 algorithm using the hybrid CSP solver.

Figure 4.9 compares overall design performance using the in-house CSP to that using the hybrid CSP solver on cases involving pattern constraints. The data for the default cases is the same as in Figure 4.8. The other series correspond to preventing the patterns `AAAA`, `CCCC`, `GGGG`, `UUUU`, `RRRRRR`, `YYYYYY`, `MMMMMM`, `KKKKKK`, `WWWWWW`, and `SSSSSS`, either through a hard pattern constraint, a hard diversity constraints, or a soft pattern constraint with one of two different weights. First, consider the second panels for each row, showing the degree of violation of the pattern constraints. We see immediately this is a different regime from the composition constraint: in the default series there is between 5-10% normalized degree of violation. Also, there appears to be no difference in these distributions for the two different CSP solvers. From this we can expect that there will be additional design cost due to imposing the constraints. When examining the results for the in-house solver, we see again the lack of approach to the 4/3 bound. However, the design cost reflects the stringency of the applied constraints, with the hard constraint having the highest cost and the lowest-weighted soft constraint having the lowest cost. Also, the two soft constraints reduce the degree of violation of the pattern constraint in the final sequences such that the higher weighted soft constraint leads to a lower degree of violation, as desired. Looking then at the results from the hybrid solver, we see that the design costs of all series have shifted lower relative to the in-house solver results. There is a different behavior for relative design cost: rather than there being a strict tradeoff between degree of violation and design cost, we are able to achieve perfect satisfaction of the constraint using the diversity constraint implementation at the same cost as when using the soft constraints. The pattern constraint implementation continues to be more costly than the other series, but less so relative to the in-house solver results.

4.4.6 Test Tube Design Performance

We now consider the benchmark results for test tube design of the engineered test set. The results for using the NUPACK 4.0 design algorithm on this test set are shown in Figure 4.10. These results are directly comparable to those in Figure B.21. For design quality, the algorithm continues to meet the 1% stop condition. In terms of raw design performance, we see speedups of up to 10×, with some of the slowest trials for the 400 nt cases approaching the results in Figure B.21. Relative design cost has increased relative to the 3.2 algorithm, pointing to the increased effect of non-thermodynamic algorithm overhead.

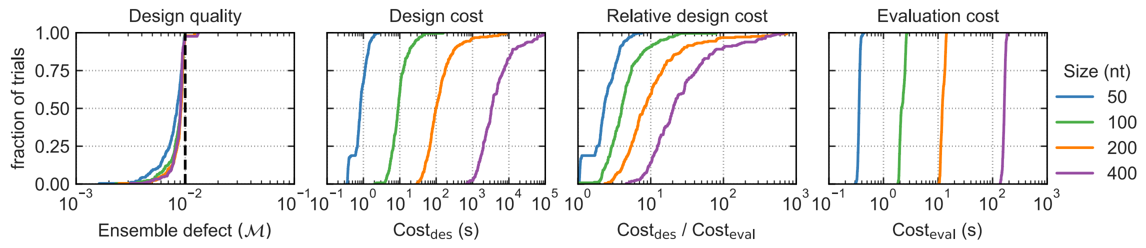


Figure 4.10: Performance of the NUPACK 4.0 design algorithm on the engineered test tube test set. First panel: Design quality. Second panel: Design cost. Third panel: Cost of sequence design relative to a single evaluation of the objective function. Fourth panel: Cost of a single evaluation of the objective function.

4.4.7 Reaction Pathway Case Studies

We now examine the performance of algorithm on reaction pathways from the nucleic acid design literature. We continue to use the five reaction pathway case studies described in Section 3.4.1.

4.4.8 Orthogonal Reaction Pathway Design

We begin by examining the performance as a function of increasing numbers of systems. The results shown in Figure 4.11 were generated by running the 4.0 algorithm over the same set of design specifications used to produce Figure B.11. The design algorithm reduced the objective function beneath $f_{\text{stop}} = 0.02$ for all trials. Additionally, we continue to see approximately an order of magnitude increase in design cost as the number of orthogonal systems doubles. However, this is moderated by an order of magnitude decrease in design time relative to the equivalent trials using the 3.2 algorithm. Thus, we can expect to be able to design twice as many orthogonal systems with the 4.0 algorithm as the 3.2 algorithm in the same amount of time. Relative design cost continues to increase with the number of systems, reflecting the increased effort spent on designing against off-targets. Relative to the 3.2 results, relative design cost is approximately the same.

We also tracked statistics on the number of times the design algorithm made use of its important subroutines. These results are shown in Figure 4.12. These data are consistent with the idea that more difficult designs, in this case corresponding to more orthogonal systems, require more frequent uses of the mechanisms of mutation, reseeding, redecomposing, and refocusing to achieve proportionately high quality designs. Indeed, we see the distributions shifting rightward as the number of orthogonal systems increases. These results also help to explain the variations we see in the results in Figure 4.11 across reaction pathways.

4.4.9 Reaction Pathway Design with Hard Constraints

We now turn to performance results for constrained design of reaction pathways. These results were generated using the constraint sets imposed on the results in Figure B.19. The results for the 4.0 algorithm are shown in Figure 4.13. We see that relative to the 3.2 results, many of the design costs have shifted downward approximately an order of magnitude. Some have shifted downward in cost at the low end of the distribution, but not the high end, indicating perhaps that these design trajectories are dominated by non-thermodynamic overhead. Also of note, the “diversity” series, which has the same set of feasible sequences as the “pattern, hard” series, achieves a lower design cost. There are also some series which have become more expensive, namely the “all, hard” series for the Boolean logic AND using toehold sequestration gates and window constraints for conditional self-assembly via HCR. The reason for this degradation in time is not understood. Additionally, the quality of the window constraints test set seems to have decreased slightly in some cases, relative to the 3.2 results.

Figure 4.14 shows the measurements of subroutine use during the design trials. Unlike Figure 4.12, where all components of the design were used more frequently in a monotonically increasing fashion with number of orthogonal systems, we cannot make such broad observations based on

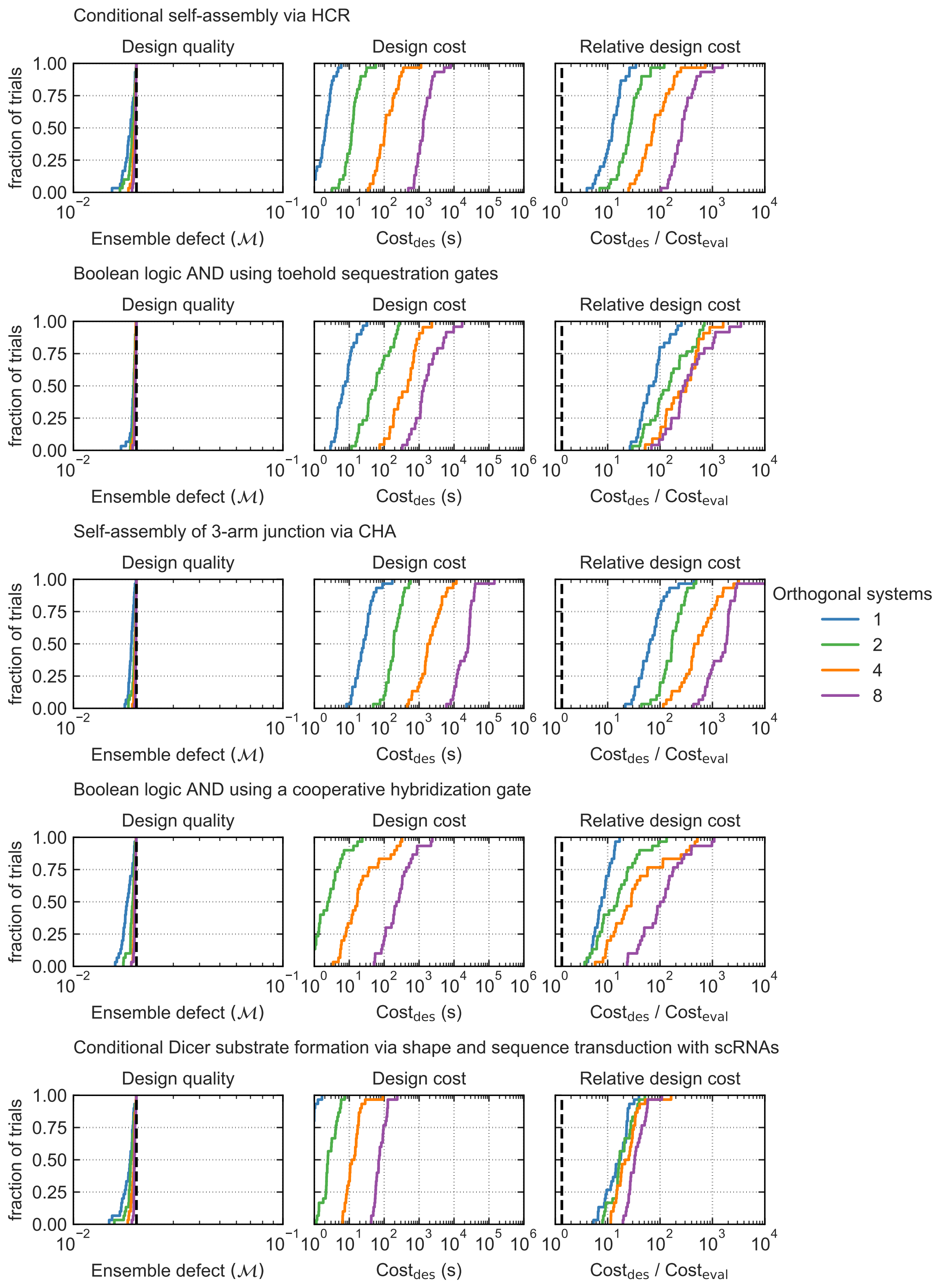


Figure 4.11: Reaction pathway design performance for 1, 2, 4, or 8 orthogonal systems. Left: Design quality. Middle: Design cost. Right: Cost of sequence design relative to a single evaluation of the objective function.

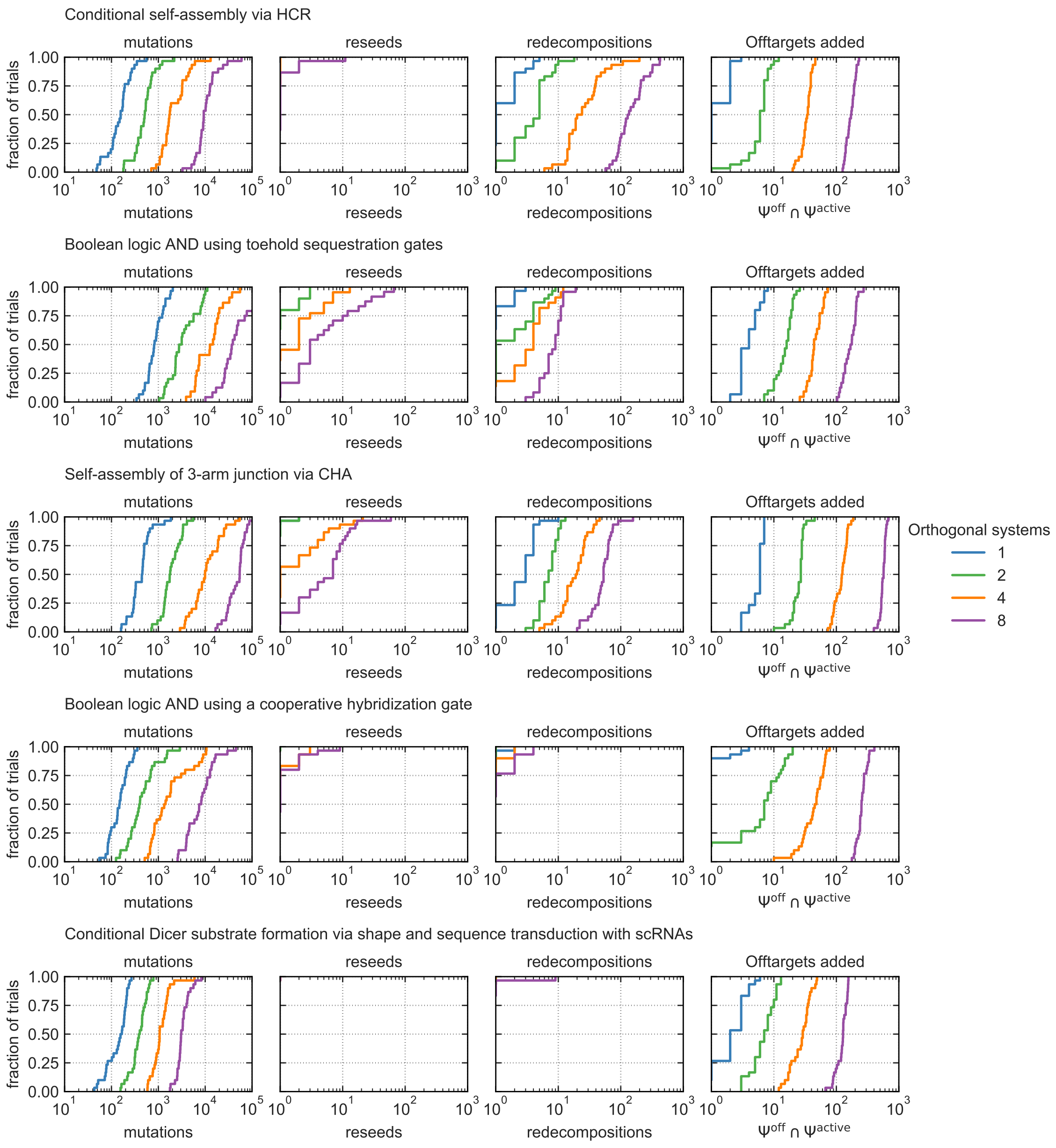


Figure 4.12: Algorithm statistics for orthogonal reaction pathway design. First panel: Number of mutations during design. Second panel: Number of times sequences were reseeded during design. Third panel: Number of times subsequence merging stopped and complexes were redecomposed during design. Fourth panel: Total number of off-targets moved from Ψ^{passive} to Ψ^{active} during refocusing in the design

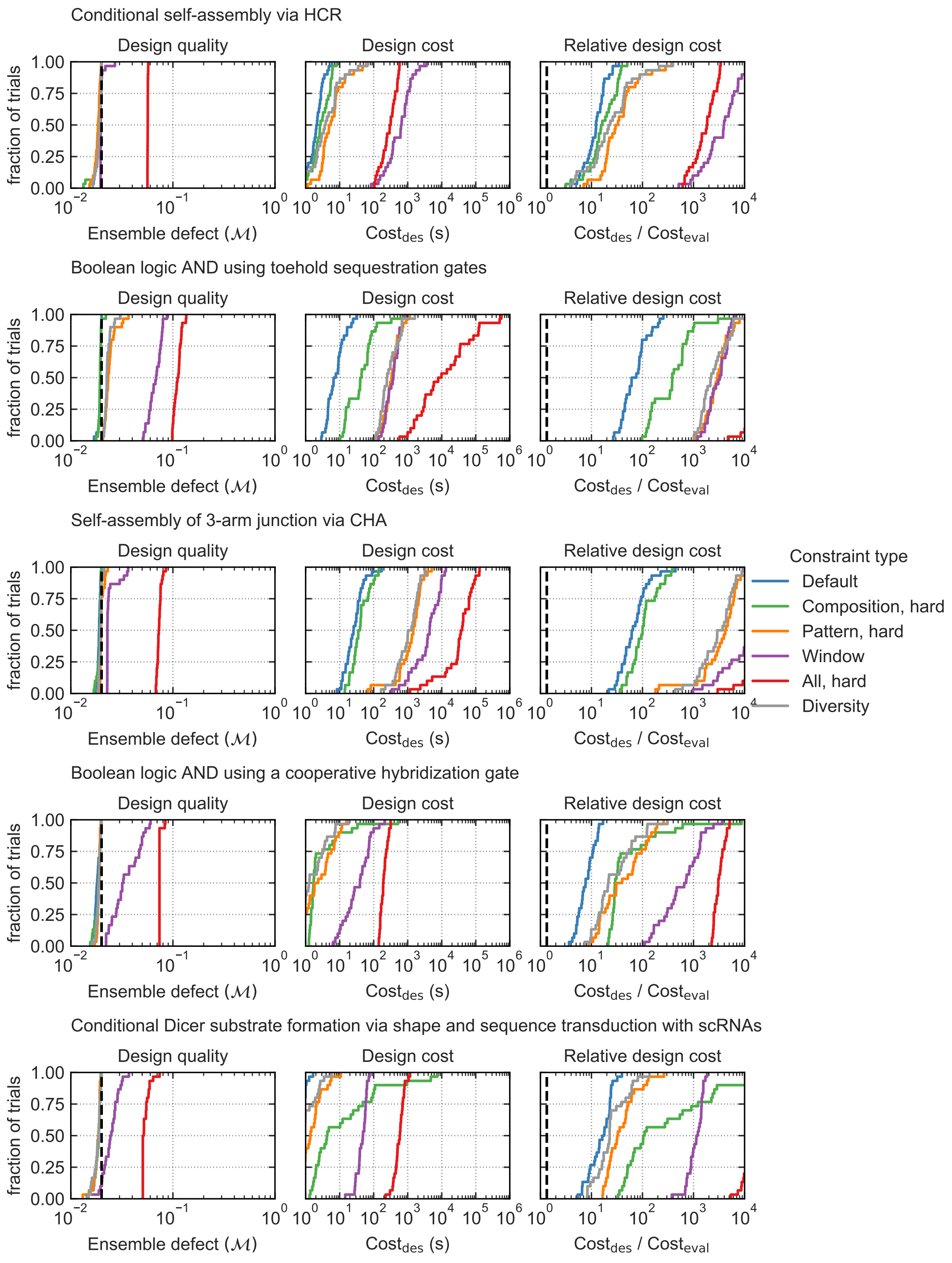


Figure 4.13: Reaction pathway design performance with sequence constraints. Left: Design quality. Middle: Design cost. Right: Cost of sequence design relative to a single evaluation of the objective function. Axes limits have been explicitly chosen to allow direct comparison with Figure B.19. Some series are partially occluded as a result. Additionally, the "default" series for the Boolean logic AND using a cooperative hybridization gate case study is not visible because all trials finish in less that 1 second.

these additional data. For instance, in Figure 4.13, the all constraints set almost always has the most expensive design cost. However, in terms of mutations, reseeds, and redecompositions, it almost always has less of these than the window constraint set. There are several possible explanations for this. First, a design trajectory with fewer redecompositions occurring close to the root will be more expensive than one with many redecompositions happening nearer to the leaves. We do not account for these differences in the data in Figure 4.14. The same can be said for refocusing more total times with fewer off-targets transfered per refocus, as each refocus corresponds to a root evaluation. Another possibility, based on the results in Section 4.4.4, is that each mutation is more costly. If during the trials with the "all, hard" constraint set the hybrid CSP solver ends up falling back on the in-house solver, then this will lead to more expensive mutations. Additionally, we expect multiple constraint types to cause the hamming distance between successive mutations to increase, especially when window constraints are present. This will lead to many more leaves being evaluated as their sequences change compared to a less stringent constraint set that changes fewer variables on average per mutation. A final contributing factor to more expensive mutations is if they occur after refocusing, when there are more nodes in the evaluation forest. It seems unlikely that a single summary statistic will capture these differences in all cases and instead individual design trajectories can be examined in detail with the software described in Appendix C.

4.4.10 Reaction Pathway Design with Hard and Soft Constraints

We now compare the performance of using hard constraints with their soft constraint counterparts. The results from Figure 4.13 are reproduced here for comparison (with series having the same colors as in that figure). In addition we show results for soft constraint versions of the composition and pattern constraints. In each of these cases, the soft constraint has weight $w_k = 0.25$. Finally, the constraint set "All, soft" uses both the preceeding soft composition constraint and soft pattern constraint along with the hard window constraint, with each soft constraint having $w_k = 0.25$. Per soft constraint type, all relevant strand sequences are within a single soft constraint scope, so that pathways with different numbers of strands still have the same number of soft constraints specified. In practice, different weights could be specified for each strand to more precisely modulate the objective function.

These results are shown in Figure 4.15. In addition to the information types shown in Figures 4.11 and 4.13, we also show the average normalized degree of violation for the soft constraints if relevant to a series. We include these values for the default case as well. Focusing on these values, we notice first that adding soft constraints to the design does lead to a reduction of the corresponding degree of violation relative to the default case. Typically this reduction is at least an order of magnitude for the "all, soft" series and more for the single soft constraint series. In the comparison between the hard and soft composition and pattern constraints, there is not a clear winner in terms of design cost. Most of the time they are approximately the same distribution and there are cases where hard dominates soft and soft dominates hard. Comparing the hard and soft versions of the all constraint set, we do see a striking difference in performance, with the soft constraint version costing 5-100×

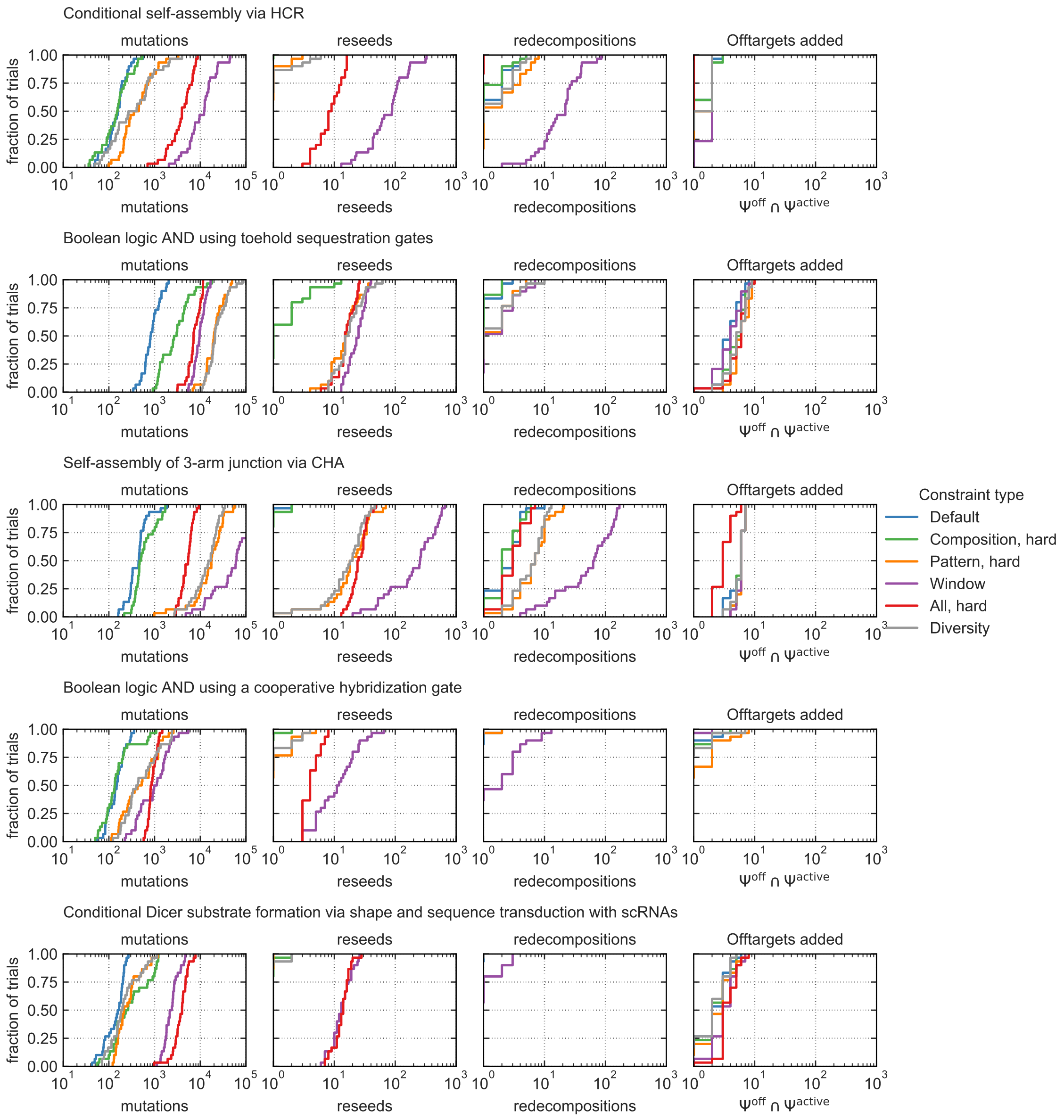


Figure 4.14: Algorithm statistics for reaction pathway design with hard constraints. First panel: Number of mutations during design. Second panel: Number of times sequences were reseeded during design. Third panel: Number of times subsequence merging stopped and complexes were redecomposed during design. Fourth panel: Total number of off-targets moved from Ψ^{passive} to Ψ^{active} during refocusing in the design

less in design cost and typically producing at least as good sequences as ranked by $\mathcal{M}$. Thus, soft constraints are expected to significantly increase design throughput on problems that do not require perfect adherence to the hard constraints.

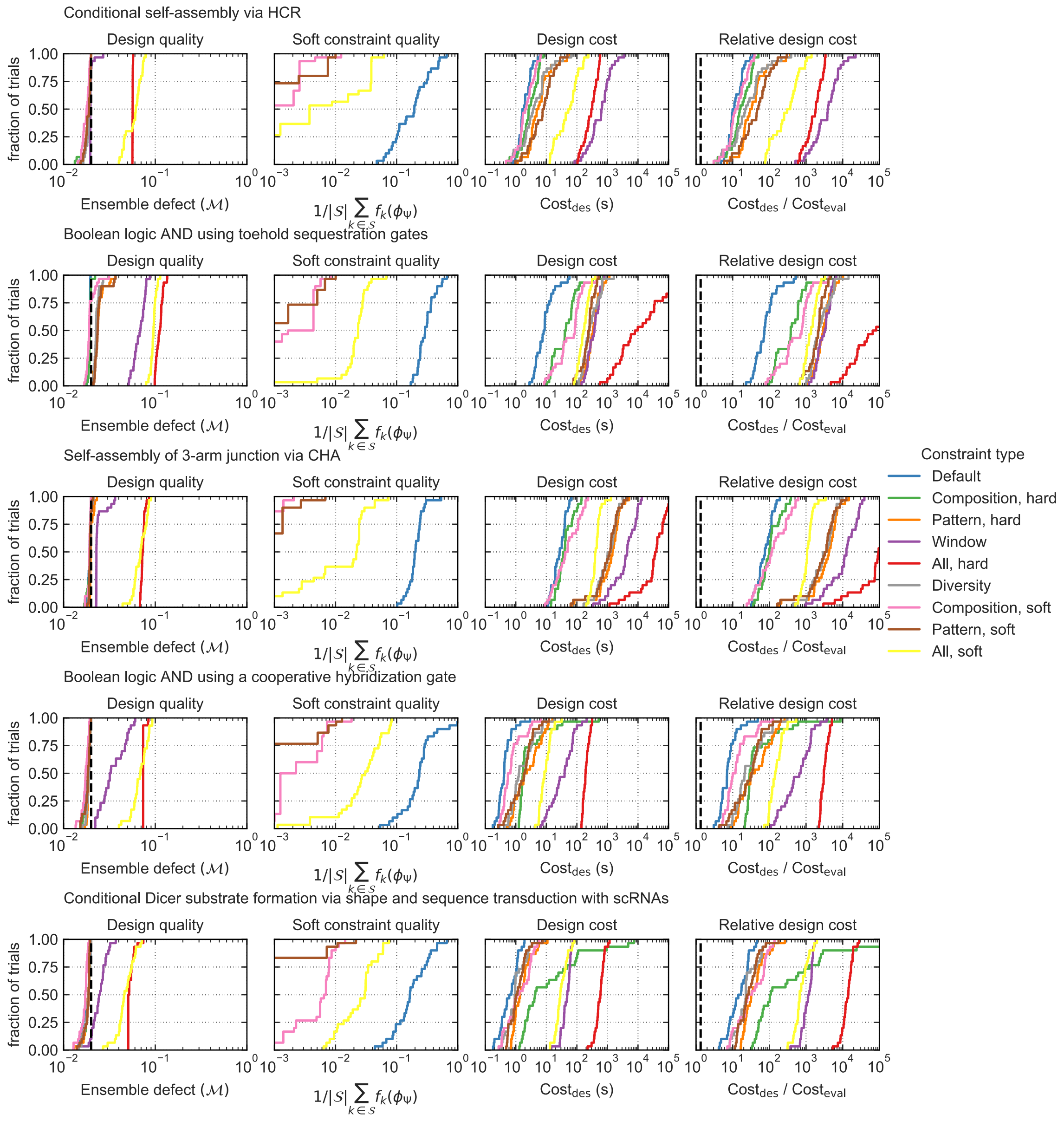


Figure 4.15: Reaction pathway design performance with hard and soft sequence constraints. First panel: Thermodynamic design quality. Second panel: Quality of soft constraints. Third panel: Design cost. Fourth panel: Cost of sequence design relative to a single evaluation of the objective function.

Figure 4.16 shows the measurements of subroutine use during the design trials with soft and hard constraints. The results include those of Figure 4.14. The discussion is similar as well: there is not a strict correlation with the quality and cost performance metrics of Figure 4.15 and these subroutine metrics. However, we can now compare between soft and hard constraints. For instance, the “all, soft” series sometimes has fewer invocations of subroutines than the “all, hard” series. Frequently,

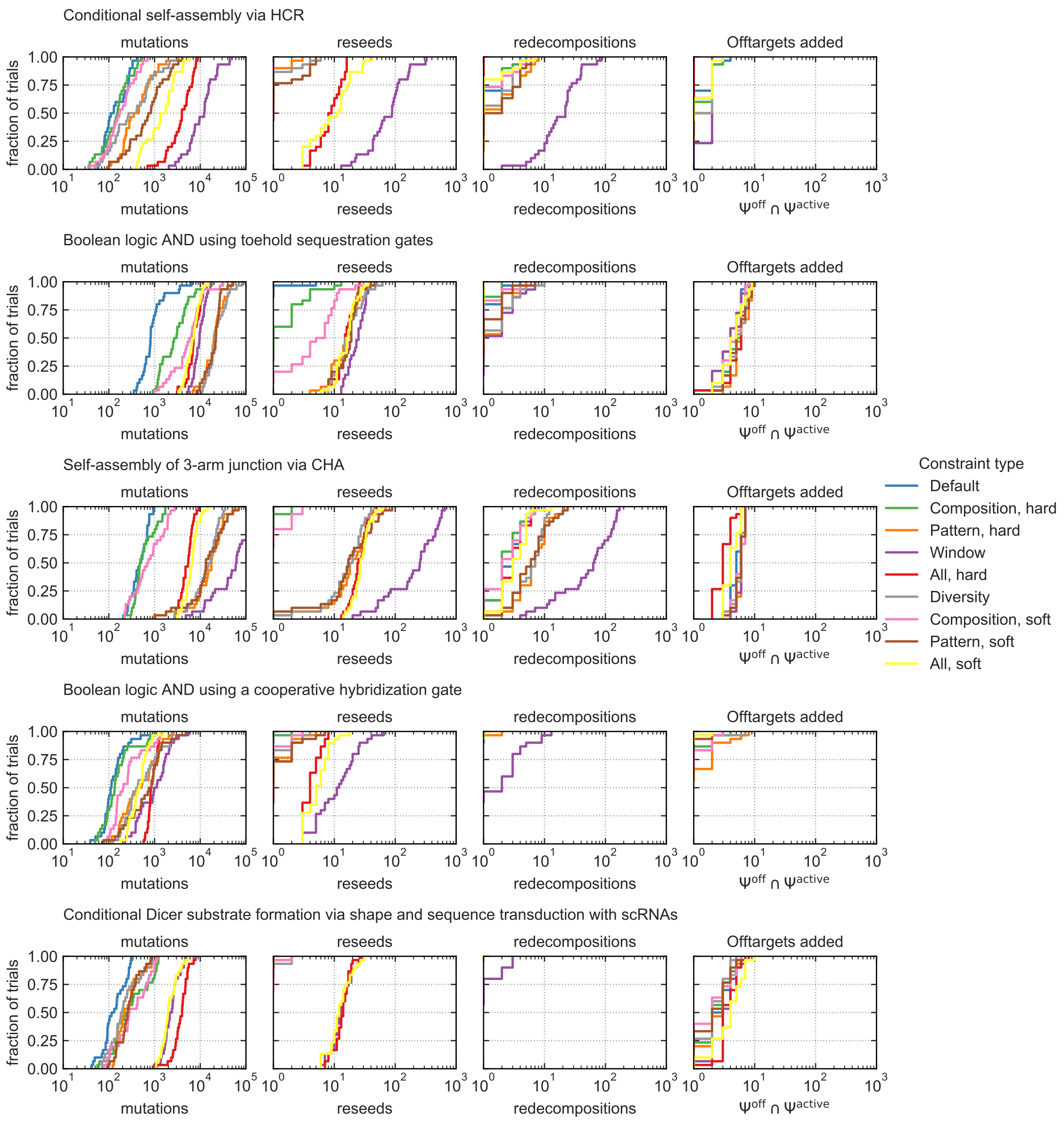


Figure 4.16: Algorithm statistics for reaction pathway design with hard and soft constraints. First panel: Number of mutations during design. Second panel: Number of times sequences were reseeded during design. Third panel: Number of times subsequence merging stopped and complexes were redecomposed during design. Fourth panel: Total number of off-targets moved from Ψ^{passive} to Ψ^{active} during refocusing in the design.

however, the distributions are very nearly identical. There is also sometimes an inversion where “all, hard” uses fewer invocations than “all, soft” This seems to corroborate the hypothesis that it is the cost per mutation of the “all, hard” constraint set that is leading to the large decreases in design cost we see when switching to the “all, soft” set.

While the foregoing results for soft-constraints have focused on comparisons of soft constraint types with analogous hard constraints, we also investigated a soft constraint without an analogue: the structure free energy equalization soft constraint. To do so, for each reaction pathway case study, we began with the specification for designing 8 orthogonal systems from Section 4.4.8. To this, we added structure free energy equalization soft constraints for important toeholds *across* the pathways. For example, this involves all domains a and separately all domains c for the HCR mechanism across the 8 pathways (see Figure B.1). Specifically, the free energies for duplex structures with each toehold and its reverse complement as their sequences are constrained to approach the median across all 8 orthogonal pathways, i.e. not a fixed external reference free energy. The results for two trials are shown in Figure 4.17. We can see that the variances of the distributions with the soft-constraint present are lower than those without. Thus structure free energy equalization does not generally come for free by minimizing $\mathcal{M}$ and can be improved through applying the soft constraint.

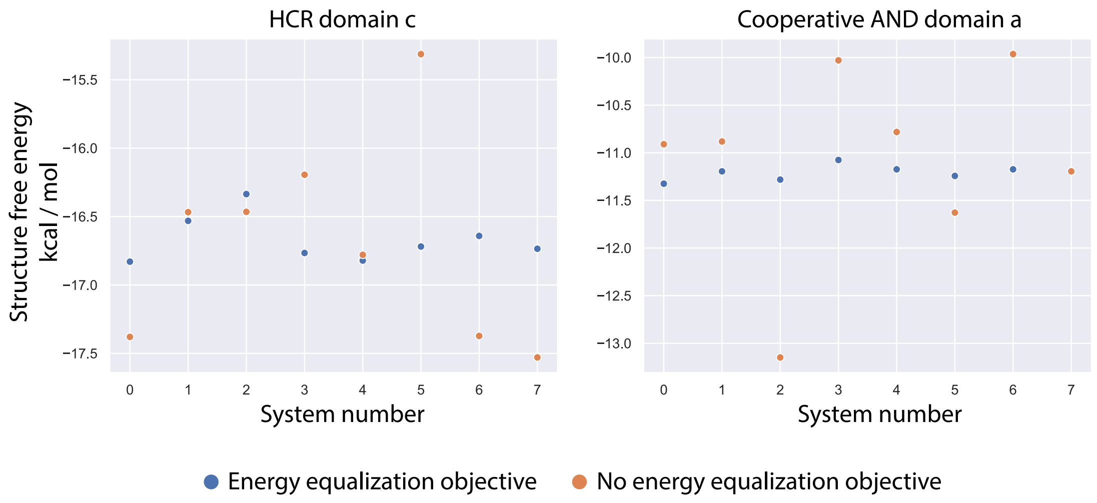


Figure 4.17: Toehold structure free energy equalization from a single trial each of two separate reaction pathways relative to the trials for the default case. The structure free energy for the duplex of the named toehold

4.4.11 Importance of Negative Design in Reducing Crosstalk

As a final performance comparison, we reproduced the methodology from Section B.2.4, using our new algorithm. The results are shown in Figure 4.18. While there are speedups for both sets of designs (with and without off-targets), the same main message holds. There is no overlap between the design quality distributions for sequences designed with and without off-target considerations; negative design against off-target complexes is necessary for high quality sequences.

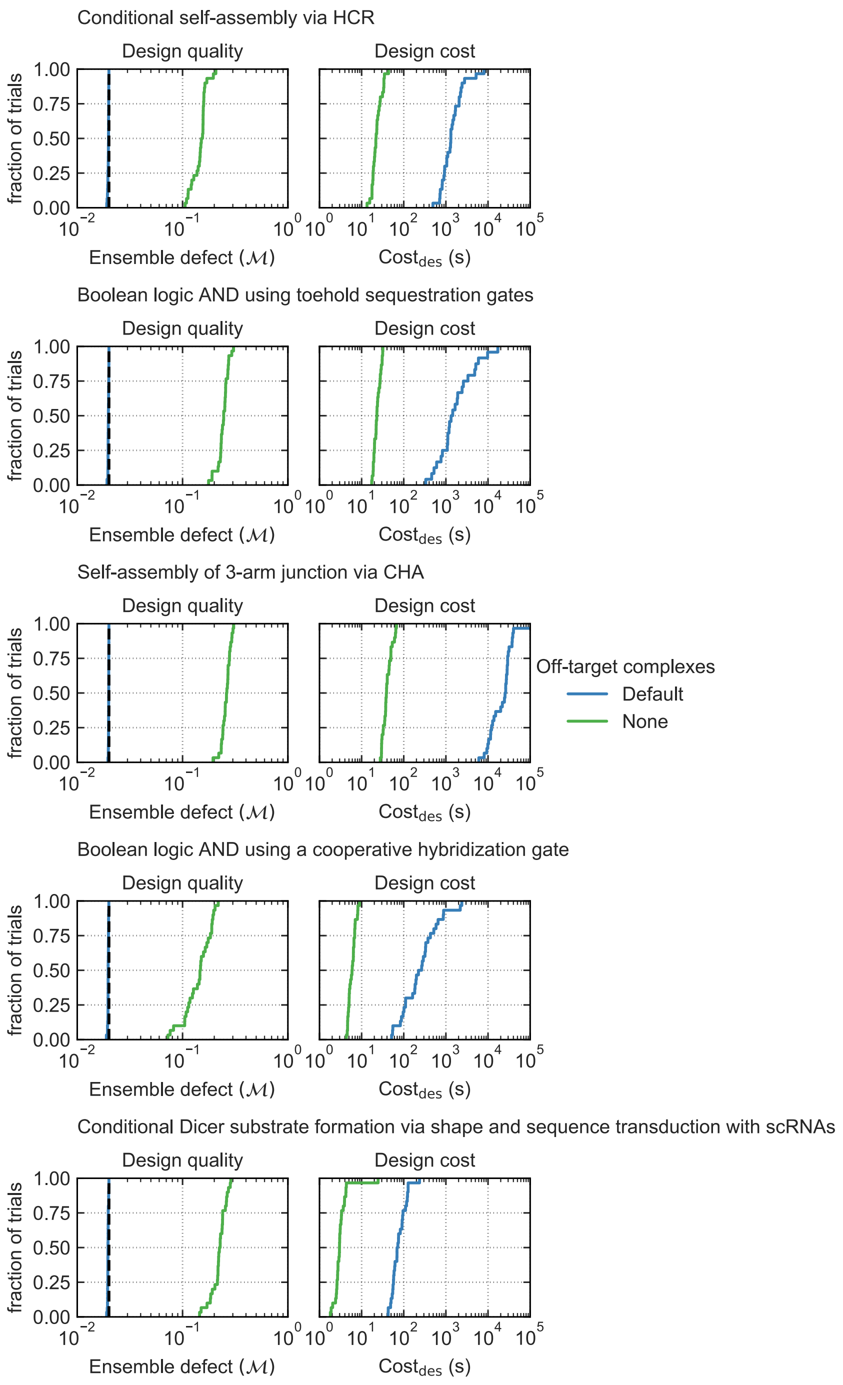


Figure 4.18: Importance of negative design in reducing crosstalk (N = 8 orthogonal systems). Comparison of designs performed with or without off-targets in the design ensemble. Left: Design quality evaluated by calculating the multistate test tube ensemble defect ($\mathcal{M}$) over the ensemble containing off-targets. Right: Design cost.

4.4.12 Implementation

The design algorithm of this chapter is a complete reimplementation of the algorithm, excluding reuse of the in-house constraint solver code of Chapter 3. The core design algorithm is written in the C++17 programming language. It uses the code of Chapter 2 to evaluate all thermodynamic quantities. We use the Gecode constraint solver library[25] as one half of our hybrid CSP solver algorithm. A Python API is provided for specifying and running designs, replacing the custom scripting language of Chapter 3. Design specifications and compoments of running designs are serialized in and out of C++ using the JSON for Modern C++ library[32]

4.5 Future Directions

The set of soft constraints discussed in Section 4.2.4 is by no means exhaustive. In fact, there are two additional soft constraints that are related to the structure free energy equalization constraints. First, we can imagine a soft constraint that equalizes the *complex free energy*, $\Delta G(\phi_i)$, across a set of sequences $\{\phi_i\}$. Second, we can extend this reasoning to equalize the *free energy differences* between sets of complexes, or,

$$\Delta\Delta G_i = \left(\sum_{\phi_e \in \phi_{e_i}} \Delta G(\phi_e)\right) - \left(\sum_{\phi_b \in \phi_{b_i}} \Delta G(\phi_b)\right),$$

where ϕ_{b_i} and ϕ_{e_i} are sets of sequences at the beginning and end of a state change. The second of these types of soft constraints would more accurately equalize toehold binding energies, which are only approximated by their most stable secondary structure with the structure free energy equalization constraint.

While such soft constraints are desirable, adding them to the design framework has more implications than the soft constraints already present. This stems from the costs of computing these quantities, $\Delta G(\phi_i)$ and $\Delta\Delta G_i$, scaling as $O(N^3)$ with the length N of the constrained sequences. This contrasts with the existing soft constraints which cost $O(N)$ to evaluate. To improve naive performance, we would likely use the hierarchical ensemble decomposition and merging paradigms to create estimates for these soft constraints during design. As the sequences involved are not necessarily on-targets or off-targets present elsewhere, this decomposition would require an initial root level evaluation to decompose the complexes. It would also necessitate changing how redecomposition works; it is currently able to ignore the soft constraints because they do not change with depth in the decomposition forest. Both of these soft constraints share the problem with the structure free energy equalization soft constraint that assignment of defect to specific sequence elements is at a very coarse level. Thus, the designer is able to tell which set of sequences i is worst, but which nucleotides within those sequences are mutated is chosen uniformly at random. This is expected to lead to more mutations on average, similar to the case of using uniform random sampling during complex design by ensemble defect minimization[28]. Another potential issue for sequences not corresponding to on-targets is that we expect the need for more frequent redecomposition, due to the potential for more extreme changes in sequence when no auxiliary structure is used to inform the decomposition.

Another area for potential investigation involves approximating the Pareto optimal set of solutions. We briefly touched on this in Section 4.2.2 when describing our objective function formulation. The Pareto optimal set contains all solutions which are not dominated by any other solution[33]. Here domination means the objective vector for one solution is less than another at every paired position along the vectors. Much like we do not seek the minimal $\mathcal{M}$ sequence in our previous design algorithm, we would seek only an approximation to this set. We already encourage our users to seek such an approximation by running multiple independent trials of the same design specification. These sequences can then be filtered either computationally or experimentally. The independence of these trials, however, means that there is nothing preventing clustering of solutions in one region of sequence space. In related problems in optimization using evolutionary algorithms, it is found that active diversity promotion mechanisms are necessary to avoid algorithmic genetic drift converging to a small region of solution space[34]. Additionally, they note that finding a set of solutions that is both a diverse and close approximation to the Pareto optimal set as the number of objectives increases is a difficult and active area of study in the multiobjective combinatorial optimization (MOCO) community. Changing design to this sort of framework would require an in-depth reexamination of all components of the present algorithm from the ground up to integrate both the general properties of MOCO solvers with the idiosyncratic estimation techniques for multistate test tube design.

4.6 Conclusion

The design algorithm of this chapter sought to improve upon the performance and extend the functionality of the multistate test tube design algorithm of Chapter 3. Implementing this algorithm atop that of the thermodynamic analysis algorithms of Chapter 2 enabled increases in raw performance, as well as access to the comprehensive dangle and coaxial stacking model and robustness to overflow for the design of large complexes. The examination of our in-house constraint solver revealed a performance bottleneck which was only visible in light of improvements in thermodynamic analysis performance. This was rectified by the introduction of our meta constraint engine, which aims to maximize performance on the easy sets of constraints while minimally impacting performance when more idiosyncratic solver heuristics are necessary. The introduction of soft constraints allows for a reduction in total design cost by up to two orders of magnitude when replacing hard constraints in the most constrained multistate test tube designs. For less stringent constraints sets (only pattern constraints or only GC-content constraints), soft constraints do not measurably reduce design cost. The soft constraint framework also allows additional flexibility in design specification through simultaneous optimization of sequence symmetry as a heuristic to avoid kinetic traps and equalization of toehold free energies to equalize rates. The degree of violation of the soft constraints maps down to the underlying nucleotide variables, as with the mulitstate test tube ensemble defect, which allows efficient defect-weighted mutation sampling. Our investigation into a simpler but nominally more expensive hierarchical decomposition method showed minimal cost differences in the asymptotic limit and led to development of a useful visualization tool for decomposition trees. Taken together, these changes make sequence design for reaction pathway engineering faster, more flexible, and more physically accurate.

Bibliography

[1] L. M. Hochrein, T. J. Ge, M. Schwarzkopf, and N. A. Pierce. "Signal transduction in human cell lysate via dynamic RNA nanotechnology". In: *ACS synthetic biology* 7.12 (2018), pp. 2796–2802.

[2] M. H. Hanewich-Hollatz, Z. Chen, L. M. Hochrein, J. Huang, and N. A. Pierce. "Conditional Guide RNAs: Programmable Conditional Regulation of CRISPR/Cas Function in Bacterial and Mammalian Cells via Dynamic RNA Nanotechnology". In: *ACS Central Science* (2019).

[3] J. Chappell, A. Westbrook, M. Verosloff, and J. B. Lucks. "Computational design of small transcription activating RNAs for versatile and dynamic gene regulation". In: *Nature communications* 8.1 (2017), p. 1051.

[4] S. Pallikkuth, C. Martin, F. Farzam, J. S. Edwards, M. R. Lakin, D. S. Lidke, and K. A. Lidke. "Sequential super-resolution imaging using DNA strand displacement". In: *PloS one* 13.8 (2018), e0203291.

[5] L. Oesinghaus and F. C. Simmel. "Switching the activity of Cas12a using guide RNA strand displacement circuits". In: *Nature communications* 10.1 (2019), p. 2092.

[6] D. Zhang and E. Winfree. "Control of DNA strand displacement kinetics using toehold exchange". In: *J. Am. Chem. Soc.* 131 (2009), pp. 17303–17314.

[7] N. Srinivas, J. Parkin, G. Seelig, E. Winfree, and D. Soloveichik. "Enzyme-free nucleic acid dynamical systems". In: *Science* 358.6369 (2017), eaal2052.

[8] H. A. Simon. "Rational choice and the structure of the environment." In: *Psychological review* 63.2 (1956), p. 129.

[9] T. F. Gonzalez. *Handbook of Approximation Algorithms and Metaheuristics, Second Edition (Chapman & Hall/Crc Computer & Information Science Series)*. Chapman & Hall/CRC, 2018. ISBN: 78-1-4987-7011-8.

[10] M. Ehrgott. "A discussion of scalarization techniques for multiple objective integer programming". In: *Annals of Operations Research* 147.1 (2006), pp. 343–360. ISSN: 02545330. DOI: 10.1007/s10479-006-0074-z.

[11] N. C. Seeman. "Nucleic acid junctions and lattices". In: *Journal of Theoretical Biology* 99.2 (Nov. 1982), pp. 237–247. ISSN: 00225193. DOI: 10.1016/0022-5193(82)90002-9. URL: https://linkinghub.elsevier.com/retrieve/pii/0022519382900029.

[12] N. Seeman and N. Kallenbach. "Design of immobile nucleic acid junctions". In: *Biophysical Journal* 44.2 (Nov. 1983), pp. 201–209. ISSN: 00063495. DOI: 10.1016/S0006-3495(83)84292-1. URL: https://linkinghub.elsevier.com/retrieve/pii/S0006349583842921.

[13] N. C. Seeman. "De novo design of sequences for nucleic acid structural engineering". In: *Journal of Biomolecular Structure and Dynamics* 8.3 (1990), pp. 573–581. ISSN: 15380254. DOI: 10.1080/07391102.1990.10507829.

[14] R. M. Dirks, M. Lin, E. Winfree, and N. A. Pierce. "Paradigms for computational nucleic acid design". In: *Nucleic Acids Research* 32.4 (2004), pp. 1392–1403. ISSN: 03051048. DOI: 10.1093/nar/gkh291.

[15] D. Soloveichik, G. Seelig, and E. Winfree. “DNA as a universal substrate for chemical kinetics”. In: *Proceedings of the National Academy of Sciences* 107.12 (2010), pp. 5393–5398.

[16] M. J. Serra and D. H. Turner. “Predicting Thermodynamic Properties of RNA”. In: *Methods Enzymol.* 259 (1995), pp. 242–261.

[17] D. H. Mathews, J. Sabina, M. Zuker, and D. H. Turner. “Expanded Sequence Dependence of Thermodynamic Parameters Improves Prediction of RNA Secondary Structure”. In: *J. Mol. Biol.* 288 (1999), pp. 911–940.

[18] S. Bommarito, N. Peyret, and J. SantaLucia. “Thermodynamic Parameters for DNA Sequences with Dangling Ends”. In: *Nucleic Acids Res.* 28.9 (2000), pp. 1929–1934. ISSN: 0305-1048. DOI: DOI10.1093/nar/28.9.1929.

[19] N. Peyret. “Prediction of Nucleic Acid Hybridization: Parameters and Algorithms”. Thesis. 2000.

[20] M. Zuker. “Mfold Web Server for Nucleic Acid Folding and Hybridization Prediction”. In: *Nucleic Acids Res.* 31.13 (2003), pp. 3406–3415.

[21] D. H. Mathews. “Using an RNA secondary structure partition function to determine confidence in base pairs predicted by free energy minimization”. In: *Rna* 10.8 (2004), pp. 1178–1190.

[22] D. H. Turner and D. H. Mathews. “NNDB: the nearest neighbor parameter database for predicting stability of nucleic acid secondary structure”. In: *Nucleic Acids Res.* 38 (2010), pp. D280–D282.

[23] B. R. Wolfe, N. J. Porubsky, J. N. Zadeh, R. M. Dirks, and N. A. Pierce. “Constrained Multistate Sequence Design for Nucleic Acid Reaction Pathway Engineering”. In: *Journal of the American Chemical Society* 139.8 (2017), pp. 3134–3144. ISSN: 15205126. DOI: 10.1021/jacs.6b12693.

[24] B. R. Wolfe. “Design and Analysis of Nucleic Acid Reaction Pathways”. PhD thesis. California Institute of Technology, 2014.

[25] C. Schulte, G. Tack, and M. Z. Lagerkvist. *Modeling and programming with gecode*. 2010.

[26] R. Dechter and D. Cohen. *Constraint processing*. Morgan Kaufmann, 2003.

[27] B. R. Wolfe and N. A. Pierce. “Sequence Design for a Test Tube of Interacting Nucleic Acid Strands”. In: *ACS Synthetic Biology* (2014), p. 141020092749006. ISSN: 2161-5063. DOI: 10.1021/sb5002196. URL: http://pubs.acs.org/doi/pdfplus/10.1021/sb5002196.

[28] J. N. Zadeh, B. R. Wolfe, and N. A. Pierce. “Nucleic acid sequence design via efficient ensemble defect optimization”. In: *Journal of Computational Chemistry* 32.3 (2011), pp. 439–452. DOI: 10.1002/jcc.21633. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jcc.21633. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.21633.

[29] R. M. Dirks, J. S. Bois, J. M. Schaeffer, E. Winfree, and N. A. Pierce. “Thermodynamic Analysis of Interacting Nucleic Acid Strands”. In: *SIAM Rev.* 49.1 (2007), pp. 65–88.

[30] C. Sanderson. “Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments”. In: (2010).

[31] J. McEntyre and J. Ostell. *The NCBI Handbook [Internet]*. Bethesda, MD: National Center for Biotechnology Information (US), 2002.

[32] N. Lohmann. *JSON for Modern C++*. Aug. 2019. URL: https://github.com/nlohmann/json.

[33] B. Li, J. Li, K. Tang, and X. Yao. "Many-objective evolutionary algorithms: A survey". In: *ACM Computing Surveys (CSUR)* 48.1 (2015), p. 13.

[34] R. C. Purshouse and P. J. Fleming. "On the evolutionary optimization of many conflicting objectives". In: *IEEE Transactions on Evolutionary Computation* 11.6 (Dec. 2007), pp. 770–784. ISSN: 1089778X. DOI: 10.1109/TEVC.2007.910138. URL: http://ieeexplore.ieee.org/document/4384508/.

Appendix A

A Unified Dynamic Programming Framework for The Analysis of Interacting Nucleic Acid Strands: Supplementary Information

This appendix was adapted from material in the supplementary info of M. E. Fornace*, N. J. Porubsky* and N. A. Pierce. "A Unified Dynamic Programming Framework for the Analysis of Interacting Nucleic Acid Strands: Enhanced Models, Robustness, and Speed". In: (2019, in preparation).

A.1 Additional model details

A.1.1 Strand association penalty

Based on dimensional analysis, we define the complex concentrations x_Ψ for a test tube containing the set of complexes Ψ as mole fraction rather than molarities (see (B.1)). Therefore, we adjust the strand association penalty

$$\Delta G^{\text{assoc}} = \Delta G^{\text{assoc}}_{\text{pub}} - kT \log[\rho_{\text{H}_2\text{O}}/(1 \text{ mol/liter})] \tag{A.1}$$

where $\Delta G^{\text{assoc}}_{\text{pub}}$ is the published value for two strands associating[1] and $\rho_{\text{H}_2\text{O}}$ is the molarity of water (e.g., $\rho_{\text{H}_2\text{O}} = 55.14$ mol/liter at 37 °C).[2] The strand association penalty for a complex of L strands (see (2.1)) is then

$$(L-1)\Delta G^{\text{assoc}}. \tag{A.2}$$

A.1.2 Salt corrections for DNA complexes

The default salt conditions for RNA[3–8] and DNA[6,9–11] parameter sets are [NaCl] = 1 M. Salt corrections are available for DNA parameters[9,10,12,13] to permit calculations for sodium, potassium, and ammonium ion concentrations ($[\text{Na}^+] + [\text{K}^+] + [\text{NH}_4^+]$) in the range 0.05 to 1.0 M and magnesium ion concentrations $[\text{Mg}^{++}]$ in the range 0.0 to 0.2 M.

The free energy of a DNA duplex at 37° is augmented by

$$-0.114 \frac{N}{2} \log[\text{Na}^+], \tag{A.3}$$

for user-specified $0.05\,\text{M} \le [\text{Na}^+] \le 1.0\,\text{M}$, where N is the number of phosphates in the duplex and it is assumed that ΔH is independent of $[\text{Na}^+]$, which is valid for this salt regime[9,12]. This salt correction was derived using duplexes with 16 bp or less and the accuracy decreases as duplex length increases further[9,12]. The expression can be generalized to monovalent sodium and potassium ions[12] as well to divalent magnesium cations[10,13]:

$$-0.114 \frac{N}{2} \log\left([\text{Na}^+] + [\text{K}^+] + [\text{NH}_4^+] + 3.3\,[\text{Mg}^{++}]^{1/2}\right), \tag{A.4}$$

for user-specified for $0.05\,\mathrm{M} \le [\mathrm{Na}^+] + [\mathrm{K}^+] + [\mathrm{NH}_4^+] \le 1.0\,\mathrm{M}$ and $0.0\,\mathrm{M} \le [\mathrm{Mg}^{++}] \le 0.2\,\mathrm{M}$.

To apply this salt correction to a complex of L strands at temperature T, consider a secondary structure s containing one or more duplexes. We assume that strands are synthesized with one phosphate per base so that $N/2 = n_{\mathrm{bp}}(\phi, s)$ where N is the total number of phosphates in duplexes and $n_{\mathrm{bp}}(s)$ is the total number of base pairs in s. We further assume that ΔH is independent of cation concentration in this regime. The secondary structure free energy $\Delta G(\phi, s)$ is then augmented by

$$n_{\mathrm{bp}}(s)\Delta G^{\mathrm{salt}} \tag{A.5}$$

with

$$\Delta G^{\mathrm{salt}} = -0.114 \log\left([\mathrm{Na}^+] + [\mathrm{K}^+] + [\mathrm{NH}_4^+] + 3.3\,[\mathrm{Mg}^{++}]^{1/2}\right)\frac{T}{T_{37}} \tag{A.6}$$

for user-specified

$$0.05\,\mathrm{M} \le [\mathrm{Na}^+] + [\mathrm{K}^+] + [\mathrm{NH}_4^+] \le 1.0\,\mathrm{M}, \tag{A.7}$$

$$0.0\,\mathrm{M} \le [\mathrm{Mg}^{++}] \le 0.2\,\mathrm{M}, \tag{A.8}$$

with $T_{37} = 310.15$ K. In order to incorporate this salt correction in dynamic programs without explicitly calculating $n_{\mathrm{bp}}(s)$, note that for a complex of L strands, the total number of loops in each secondary structure is

$$n_{\mathrm{loop}}(s) = n_{\mathrm{bp}}(s) + 1. \tag{A.9}$$

This may be seen, for example, by starting with a single strand with no base pairs (corresponding to a single exterior loop). Each addition of a base pair adds one loop. Once all base pairs in s have been added, each addition of a nick increases the number of strands by one without changing the number of loops (all secondary structures in the complex ensemble are connected so introduction of each nick converts a loop from another type to an exterior loop). Let $n_{\mathrm{loop}}^{\mathrm{other}}$ denote the total number of non-exterior loops and $n_{\mathrm{loop}}^{\mathrm{exterior}}$ the total number of exterior loops, so we have

$$n_{\mathrm{loop}}(s) = n_{\mathrm{loop}}^{\mathrm{exterior}}(s) + n_{\mathrm{loop}}^{\mathrm{other}}(s). \tag{A.10}$$

For a complex of L strands, $n_{\mathrm{loop}}^{\mathrm{exterior}}(s) = L$. Thus, the salt correction (A.6) becomes

$$n_{\mathrm{bp}}(s)\Delta G^{\mathrm{salt}} = (L-1)\Delta G^{\mathrm{salt}} + n_{\mathrm{loop}}^{\mathrm{other}}(s)\Delta G^{\mathrm{salt}}. \tag{A.11}$$

Hence, the salt correction can be implemented by adding ΔG^{salt} to every $\Delta G(\mathrm{loop})$ except for exterior loops as a pre-processing step, using our suite of dynamic programs without modification, and then treating the constant term $(L-1)\Delta G^{\mathrm{salt}}$ in a post-processing step (see Section A.1.4).*

*A bulge loop of size one is a special case because it is treated in the model as the sum of two loop free energies that are used separately elsewhere in the model. Hence, for bulges of size one, one salt correction must be removed to avoid doubling the salt correction in this case.

A.1.3 Temperature dependence

The loop-based free energy model (2.1) is temperature dependent. Each loop free energy is calculated using

$$\Delta G(\text{loop}) = \Delta H(\text{loop}) - T\Delta S(\text{loop}) \tag{A.12}$$

where T is in Kelvin and $\Delta H(\text{loop})$ and $\Delta S(\text{loop})$ are assumed to be temperature independent[12]. Model parameters are provided for RNA[3–8] and DNA[6,9–11] in the form of $\Delta G_{37}(\text{loop})$ and $\Delta H(\text{loop})$ which can be used to calculate

$$\Delta S(\text{loop}) = \frac{1}{T_{37}}[\Delta H(\text{loop}) - \Delta G_{37}(\text{loop})] \tag{A.13}$$

with $T_{37} = 310.15$ K, so (A.12) becomes

$$\Delta G(\text{loop}) = \Delta H(\text{loop}) - \frac{T}{T_{37}}[\Delta H(\text{loop}) - \Delta G_{37}(\text{loop})]. \tag{A.14}$$

Similarly, for the strand association penalty (A.1):

$$\Delta G^{\text{assoc}}_{\text{pub}} = \Delta H^{\text{assoc}}_{\text{pub}} - T\Delta S^{\text{assoc}}_{\text{pub}}. \tag{A.15}$$

and the provided parameters $\Delta G^{\text{assoc}}_{37,\text{pub}}$ and $\Delta H^{\text{assoc}}_{\text{pub}}$ yield

$$\Delta G^{\text{assoc}}_{\text{pub}} = \Delta H^{\text{assoc}}_{\text{pub}} - \frac{T}{T_{37}}[\Delta H^{\text{assoc}}_{\text{pub}} - \Delta G^{\text{assoc}}_{37,\text{pub}}]. \tag{A.16}$$

The temperature dependence is explicit in the form of the symmetry correction (2.5) and salt correction (A.6).

A.1.4 Treatment of constant free energy terms for calculations on complex ensembles

- **Partition function**

$$\overline{Q}(\phi) = \exp\{-(L-1)[\Delta G^{\text{salt}} + \Delta G^{\text{assoc}}]/kT\}Q_{1,N} \tag{A.17}$$

where $Q_{1,N}$ is the partition function value returned by the dynamic program for a complex of N nucleotides.

A.2 Recursion diagrams and equations overview

Recursions are the bedrock of dynamic programming algorithms that specify the the relations between subproblems. In the case of nucleic acid structural ensembles, each recursion corresponds to an efficient iteration through a *conditional ensemble* of subtructures within a given subsequence that are compatible with a specified set of constraints. For a given recursion, a conditional ensemble might include an explicit structural element, which can be considered the base case of the recursion, or a reference to the result of another recursion.

A.2.1 Recursions across intrastrand and interstrand blocks

We begin by detailing the full set of recursions with and without coaxial and dangle stacking included. Reference 2 outlined dynamic programming algorithms within a multistranded ensemble which used recursions containing persistent, repeated checks of strand breaks next to each considered nucleotide. Such an approach yielded one set of recursions for both single and multistranded ensembles, but (1) sometimes yielded unnecessary complications and computations in the pseudocode and (2) eliminated any possibility of vectorization due to the conditional checks within every for-loop present.

In contrast, in keeping with the blockwise approach to calculation of multistranded ensembles, we developed separate sets of recursions for intrastrand and interstrand blocks (Figure A.1). In doing so, the intrastrand recursions are kept as simple as possible, while the interstrand recursions may still be efficiently vectorized.

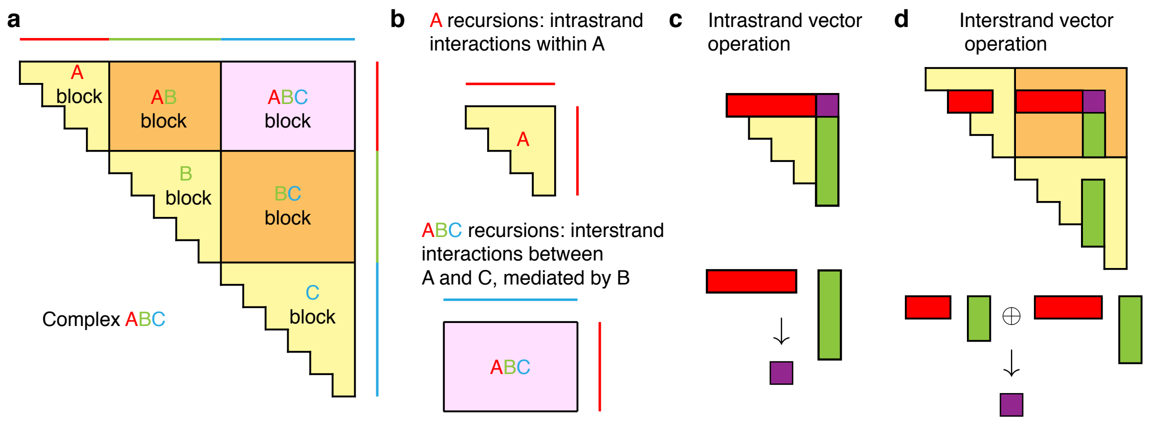

a
A block
AB block
ABC block
B block
BC block
C block
Complex ABC
b
A recursions: intrastrand interactions within A
A
ABC recursions: interstrand interactions between A and C, mediated by B
ABC
c
Intrastrand vector operation
d
Interstrand vector operation
⊕

Figure A.1: (a) Depiction of the blockwise approach. (b) Illustration of intrastrand vs interstrand recursion types. Intrastrand recursions are used to calculate triangular blocks representing conditional ensembles formed within a single strand. Interstrand recursions are used to calculate rectangular blocks representing conditional ensembles formed between two separate strands (with any number of intervening strands between them in the strand ordering). (c) Illustration of the common dot product appearing in computations of elements within an intrastrand block. Such a dot product can be straightforwardly vectorized. (d) Illustration of the common dot products appearing in computations of elements within an interstrand block. Select elements are commonly excluded from these dot products which would introduce disconnected secondary structures into the complex ensemble. These dot products can be individually vectorized with recursions that explicitly work over the strand breaks that are present.

A.2.2 Recursion conventions

In the following sections we will describe the specifics of the recursions necessary to compute thermodynamic quantities either with or without coaxial and dangle stacking. Each recursion is designed to accomplish an efficient, unique iteration through all conditional ensembles compatible with a given set of defined constraints. In addition to the change to separate interstrand and intrastrand recursions mentioned above, previous presentations[2,14] have shown $O(N^4)$ and $O(N^3)$ implementations that differ in number of matrices. Here we maintain the number of matrices between the implementations of the two complexities. The handling of interior loop contributions is the only place in the implementation that determines whether the scaling is $O(N^4)$ and $O(N^3)$ and the relevant differences are discussed in Section A.3.3.

Each recursion is represented in two ways: graphically, as a set of polymer graph recursion diagrams, and algebraically, as an equation defining the recursion as a specific combination of contributions. The polymer graph recursion diagrams have a number of general features. First, the solid circular arcs represent the backbone of the nucleic acid polymer, with the $3'$ end indicated by an arrowhead. Second, dots indicate particular nucleotide positions that define the bounds of recursive contributions. These dots are labeled with indices corresponding to the recursion equation. In cases where a dot is unlabeled, it may be assumed that the label is either $i + 1$ or $i - 1$, where i is the nearest label (indices increase from $5'$ to $3'$). Third, straight lines delimit boundaries within a given contribution. Solid straight lines indicate that the connected nucleotides are base pairing. Dashed straight lines indicate that the connected nucleotides may or may not be base pairing. Half-solid, half-dashed lines indicate that the nucleotide connected on the solid side is involved in a base pair with another nucleotide in the demarcated region. Dotted lines indicate the connected nucleotides are involved in either dangle stacking on an adjacent explicit base pair or are in two explicit base pairs that are coaxially stacking on each other. Finally, colored shading indicates that all nucleotides on the perimeter of the shaded region are contributing to an explicit energy term, whose type maps to the color.

All of the following recursion equations implicitly use one of the evaluation algebras mentioned in A.5 for the definitions of $\mathbb{0}$, $\mathbb{1}$, $\oplus$, $\otimes$, W, and Q. Due to our choice of symbols, a recursion can be understood on first reading as computing a partition function (e.g. with $\mathbb{0} \to 0, \mathbb{1} \to 1, \oplus \to +, \otimes \to \times, W(\Delta G) \to \exp(-\frac{\Delta G}{kT})$). Each equation is presented here as a formula $R_s^a(i, j, \phi)$, where a is a recursion type, s denotes whether the recursion is for an interstrand ($s = \textsc{Multi}$) or intrastrand ($s = \textsc{Single}$) block, ϕ is the sequence of the analyzed complex, and i, j are indices of ϕ. After it is evaluated for the first time, $R_s^a(i, j, \phi)$ is used to yield $Q_{i,j}^a$ in subsequent recursions.

Finally, we make clear in our pseudocode which operations may be vectorized via SIMD operations on contiguous arrays via the function DOT, which represents a dot product generalized to a variadic number of arguments, each of which is vector of a common length n:

```
DOT(vectors)
    $n \leftarrow$ LENGTH(vectors$_1$)
    $x \leftarrow \mathbb{0}$
    for $i \in [1 : n]$
        $t \leftarrow \mathbb{1}$
        for $a \in$ vectors
            $t \leftarrow t \otimes a_i$
        $x \leftarrow x \oplus t$
    return $x$
```

Algorithm A.1: Generalized dot product over multiple equal length vectors.

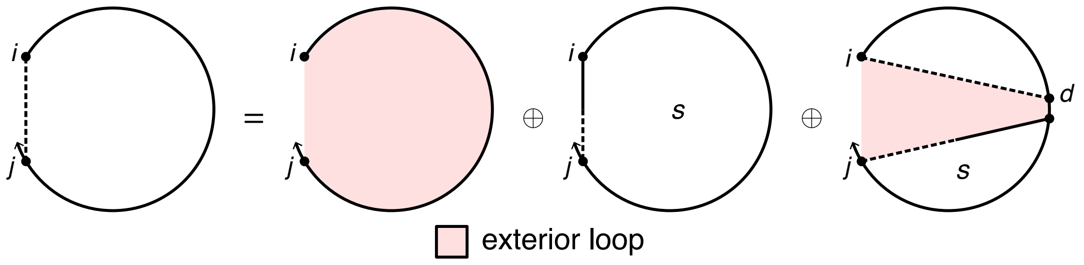
The **vectors** argument to this subroutine is frequently composed of contiguous subvectors of the rows and columns of the matrices storing the result of previous recursion evaluations, e.g. the matrix Q^b. These are denoted by replacing the typical scalar index with a range for either their row or column indices. These ranges are written such that $[i : j]$ yields $i, i+1, \ldots, j-1, j$. The ranges can also be used to create a vector of free energy values. When there are multiple ranges, the elements match with each other such that $a + b[i : j] + c[d : e]$ yields $a + bi + cd, a + b(i+1) + c(d+1), \ldots, a + b(j-1) + c(e-1), a + bj + ce$. Also, sometimes to match up the values in the vectors correctly, two ranges must proceed in opposite directions (ascending and descending). A descending, or *reversed*, range is written such that $[i : j]^r$ yields $j, j-1, \ldots, i+1, i$.

A.3 Recursions excluding coaxial stacking and dangles

The following sections describe the specific recursions derived to match the secondary structure model ignoring dangle stacking and coaxial stacking. Relative to the previous implementation[2], the two heuristics used to incorporate a subset of dangle stacking states are not discussed. These heuristic options for dangles are implemented and usable in the code to maintain backwards compatibility.

A.3.1 Intrastrand dynamic programming recursions

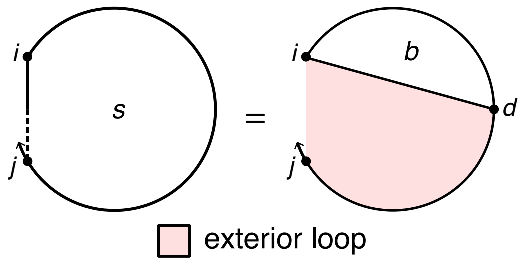
We begin with the recursion $R^{\varnothing}_{\text{SINGLE}}(i, j, \phi)$ shown in Figure A.2 and Equation A.18. In the partition function calculation, this recursion computes the unconstrained partition function for subsequence $[i, j]$. This recursion distinguishes two cases: those that have at least one top level base pair, i.e. a base pair in an exterior loop context, and the empty case where there are no base pairs in $[i, j]$. The empty case has contribution $\mathbb{1}$ because a secondary structure with no base pairs is defined to be the reference state with free energy 0. In cases with at least one top level base pair, there is a $3'$−most base pair which begins at $d+1$ and ends in the interval $[d+2, j]$ and is incorporated through $Q^s_{d+1,j}$. The subsequence $[i, d]$ is incorporated through recursive access of the dependent value $Q^{\varnothing}_{i,d}$. The limiting case where the index $d+1 = i$ is shown explicitly to indicate that no $Q^{\varnothing}$ element is accessed.



$$R^{\varnothing}_{\textsc{Single}}(i,j,\phi) \equiv \mathbb{1} \oplus \begin{cases} Q^s_{i,j} \oplus \textsc{dot}(\{Q^{\varnothing}_{i,[i:j-5]}, Q^s_{[i+1:j-4],j}\}), & j-i>4 \\ Q^s_{i,j}, & j-i=4 \\ \mathbb{0}, & \text{otherwise} \end{cases} \tag{A.18}$$

Figure A.2: Polymer graph recursion diagram and recursion equation for the intrastrand $R^{\varnothing}$ recursion.

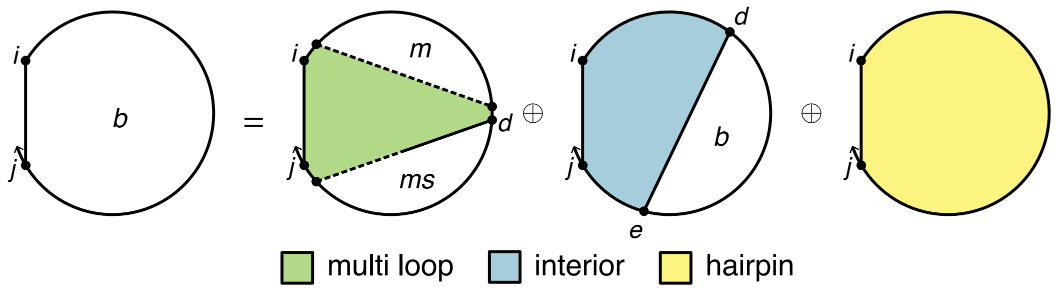
Next, elements of the type $Q^s_{i,j}$ are computed using the recursion $R^s_{\textsc{Single}}(i,j,\phi)$, shown in Figure A.3 and Equation A.19. This recursion collects contributions from all top level base pairs starting at i and ending in $[i+1,j]$. The contribution from each base pair $i \cdot d$ is incorporated through $Q^b_{i,d}$. There is no need to account for any additional energy contributions as all these states are in exterior loops with no dangle or coaxial stacking. The index limits are determined by the sterically-imposed minimum hairpin size of 3 unpaired nucleotides.



$$R^s_{\textsc{Single}}(i,j,\phi) \equiv \begin{cases} \bigoplus_{d=1+4}^{j} Q^b_{i,d}, & j-i>4 \\ Q^b_{i,j}, & j-i=4 \\ \mathbb{0}, & \text{otherwise} \end{cases} \tag{A.19}$$

Figure A.3: Polymer graph recursion diagram and recursion equation for the intrastrand R^s recursion.

Elements of the type $Q^b_{i,j}$ are computed using the recursion $R^b_{\textsc{Single}}(i,j,\phi)$, shown in Figure A.4 and Equation A.20. In the partition function calculation, this recursion computes the conditional partition function for subsequence $[i,j]$ given that i and j are base paired. The function Pairable checks if two nucleotides can form a Watson–Crick or wobble base pair based only on nucleotide type without considering sterics. In each contribution to $Q^b_{i,j}$, base pair $i \cdot j$ bounds one of three types of loop: a multiloop, and interior loop, or a hairpin loop. In the multiloop cases, there are at least two other base pairs bounding the remainder of the multiloop. We recursively and uniquely consider all multiloop cases by obtaining the contribution from all $3'$−most base pairs starting at d and ending in $[d+1, j]$ from $Q^{ms}_{d,j-1}$ and all possible other states of one or more base pairs in $[i+1, d-1]$ from $Q^m_{i+1,d-1}$. At this level of recursion, we account for the multiloop formation penalty, $\Delta G^{\text{multi}}_{\text{init}}$, as well as the branch penalty for base pair $i \cdot j$, $\Delta G^{\text{multi}}_{\text{bp}}$. The multiloop cases need only be considered when it is sterically possible for two additional base pairs to form in $[i+1, j-1]$. For the interior loop case, we defer discussion until Section A.3.3 where the two alternatives Single $O(N^4)$ Interior(i, j, ϕ) and Single $O(N^3)$ Interior(i, j, ϕ) are discussed. Finally, for the hairpin case, as long as i and j exceed the minimum hairpin size, there is one contribution with recursion energy equal to hairpin energy $\Delta G^{\text{hairpin}}_{i,j}(\phi)$.

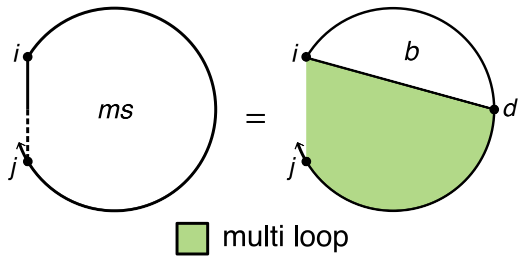


$$R^b_{\text{Single}}(i, j, \phi) \equiv \begin{cases} C_1, & \text{Pairable}(\phi_i, \phi_j) \\ \mathbb{0}, & \text{otherwise} \end{cases}$$

$$\text{where } C_1 \equiv \begin{cases} \text{dot}(\{Q^m_{i+1,[i+5:j-6]}, Q^{ms}_{[i+6:j-5],j-1}\}) \otimes W(\Delta G^{\text{multi}}_{\text{init}} + \Delta G^{\text{multi}}_{\text{bp}}), & j-i > 10 \\ \mathbb{0}, & \text{otherwise} \end{cases}$$
$$\oplus \text{ Single Interior}(i, j, \phi)$$
$$\oplus W(\Delta G^{\text{hairpin}}_{i,j}(\phi)) \tag{A.20}$$

Figure A.4: Polymer graph recursion diagram and recursion equation for the intrastrand R^b recursion.

Analogous to the recursion $R^s_{\text{Single}}(i, j, \phi)$, $R^{ms}_{\text{Single}}(i, j, \phi)$, shown in Figure A.5 and Equation A.21, collects contributions from all base pairs starting at i and ending in $[i+1, j]$. However, in this recursion, each base pair and set of unpaired bases are in a multiloop context. We obtain the contribution of the base-paired region from $Q^b_{i,d}$. This is combined with the branch penalty for base pair $i \cdot d$, $\Delta G^{\text{multi}}_{\text{bp}}$, as well as the contribution from the unpaired bases, $\Delta G^{\text{multi}}_{\text{nt}}(j-d)$. Note that in the dot product the range multiplying $\Delta G^{\text{multi}}_{\text{nt}}$ runs in reverse order because $j-d$ decreases in size as $d-i$ increases in size.

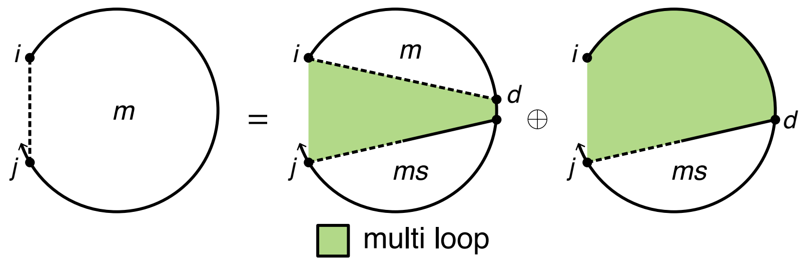


$$R^{ms}_{\textsc{Single}}(i,j,\phi) \equiv \begin{cases} Q^b_{i,j} \otimes W(\Delta G^{\text{multi}}_{\text{bp}}) \oplus C_1, & j-i>4 \\ Q^b_{i,j} \otimes W(\Delta G^{\text{multi}}_{\text{bp}}), & j-i=4 \\ \mathbb{0}, & \text{otherwise} \end{cases} \tag{A.21}$$

$$\text{where } C_1 \equiv \textsc{dot}(\{Q^b_{i,[i+4:j-1]}, W(\Delta G^{\text{multi}}_{\text{bp}} + \Delta G^{\text{multi}}_{\text{nt}}[1:j-i-4]^r)\})$$

Figure A.5: Polymer graph recursion diagram and recursion equation for the intrastrand R^{ms} recursion.

To finish the discussion of multiloop contributions, we turn to recursion $R^m_{\textsc{Single}}(i,j,\phi)$, shown in Figure A.6 and Equation A.22. There are two types of contributions: those with exactly one base pair and those with more than one base pair. The single base pair case is handled by collecting all states with base pairs starting at d and ending in $[d+1,j]$ through $Q^{ms}_{d,j}$. As the R^{ms} recursion handles the penalty for the base pairs it accumulates, only the penalty for the remaining unpaired bases are considered here: $\Delta G^{\text{multi}}_{\text{nt}}(d-i)$. The multiple base pair case is handled by obtaining information about the $3'$−most base pair through $Q^{ms}_{d+1,j}$ and the remaining subsequence is handled by a recursive access of the dependent value $Q^m_{i,d}$. In this case, the subsidiary recursions have handled all explicit multiloop energies.



$$R^{m}_{\textsc{Single}}(i,j,\phi) \equiv \begin{cases} Q^{ms}_{i,j} \oplus C_1, & j-i>4 \\ Q^{ms}_{i,j}, & j-i=4 \\ \mathbb{0}, & \text{otherwise} \end{cases} \tag{A.22}$$

$$\begin{aligned} \text{where } C_1 \equiv\ & \textsc{dot}(\{Q^{m}_{i,[i:j-5]}, Q^{ms}_{[i+1:j-4],j}\}) \\ & \oplus \textsc{dot}(\{Q^{ms}_{[i+1:j-4],j}, W(\Delta G^{\text{multi}}_{\text{nt}}[1:j-i-4])\}) \end{aligned}$$

Figure A.6: Polymer graph recursion diagram and recursion equation for the intrastrand R^m recursion.

A.3.2 Interstrand dynamic programming recursions

A frequent type of dot product that appears in intrastrand recursions has the form

$$\textsc{dot}(\{M_{i,[a:b]}, N_{[a+1:b+1],j}\}).$$

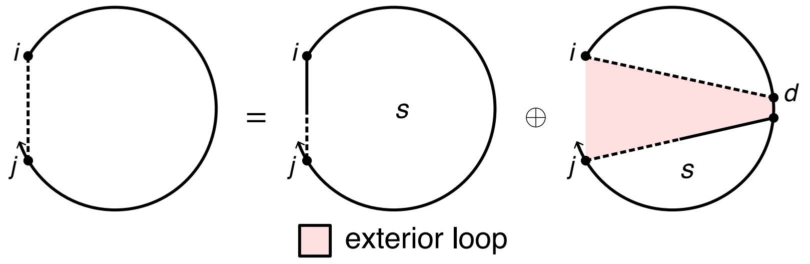
In words, it is the dot product of a sub-row of one matrix with a sub-column of a second matrix where the column indices of the first are one less than the matching row indices of the second. In interstrand blocks, the analogous types of contribution are more complicated because ranges of elements that are multiplied together typically must be on the same strand, either to ensure the complex is connected or to avoid exterior loops in the wrong contexts. Therefore, we need a method for finding the equivalent sub-rows and sub-columns for interstrand blocks. The following subroutine, VP standing for *valid positions*, does exactly that. It returns two arrays $\mathbf{L}$ and $\mathbf{R}$, each of which contain the same number of arrays of indices. Thus, for instance, the first element of $\mathbf{L}$, $\mathbf{L}_1$, is an array of indices. For a given index a, the two arrays of indices $\mathbf{L}_a$ and $\mathbf{R}_a$ are equivalent to $[a:b]$ and $[a+1:b+1]$ in the above example. Each pair $\mathbf{L}_a$ and $\mathbf{R}_a$ corresponds to an equal length pair of red sub-row and green sub-column in the bottom of Figure A.1 (d). Each index a into $\mathbf{L}$ and $\mathbf{R}$ corresponds to a region of a single strand. There is at most one such index per strand, and there may be none for a strand that is too short or for the first or last strand if i or j, respectively, is too close to a strand break.

```
VP($i, j, \phi$)
    $\mathbf{L} \leftarrow []$
    $\mathbf{R} \leftarrow []$
    $b \leftarrow \eta$ // list of base indices following strand breaks; ascending order
    $m \leftarrow$ First($b$)
    $n \leftarrow$ Last($b$)
    if $i + 1 < m$ and $j \geq n$
        Append($\mathbf{L}$, $[i, m - 2]$)
        Append($\mathbf{R}$, $[i + 1, m - 1]$)
    if $i < m$ and $j - 1 \geq n$
        Append($\mathbf{L}$, $[n, j - 1]$)
        Append($\mathbf{R}$, $[n + 1, j]$)
    if $i < m$ and $j \geq n$
        for $d \in [1 : \text{Length}(b) - 1]$
            if $b_{d+1} - b_d > 1$
                Append($\mathbf{L}$, $[b_d, b_{d+1} - 2]$)
                Append($\mathbf{R}$, $[b_d + 1, b_{d+1} - 1]$)
    return $\mathbf{L}$, $\mathbf{R}$
```

Algorithm A.2: Enumerate valid positions for vectorization in interstrand blocks.

The symbol η referenced in VP is used differently than the description of the multistranded partition function algorithms in Reference 2, where it was a function used to count the number of nicks in a given subsequence. Here, η is an array of the indices of the $5'$−most nucleotide of each strand in the interstrand block being considered, which are the positions following nicks. It is used in the interstrand recursions to ensure that states are not disconnected and that exterior loops only appear when they are being explicitly considered. This is one of the main innovations of the interstrand/intrastrand recursion split: considering only those indices that are on valid strands for the given contribution using η.

For the interstrand recursions, we begin with $R^{\varnothing}_{\text{Multi}}(i, j, \phi)$ shown in Figure A.7 and Equation A.23. In the partition function calculation, this recursion computes the unconstrained partition function for subsequence $[i, j]$. Unlike $R^{\varnothing}_{\text{Single}}(i, j, \phi)$, there is no empty case because this would correspond to a disconnected structure, which is not in the multistranded ensemble. Thus, there is only one contribution type: cases having at least one top level base pair, i.e. a base pair in an exterior loop context. In these cases, there is a $3'$−most base pair which begins at $d + 1$ and ends in the interval $[d + 2, j]$ and is incorporated through $Q^{s}_{d+1,j}$. The subsequence $[i, d]$ is incorporated through recursive access of the dependent value $Q^{\varnothing}_{i,d}$. The limiting case where the index $d + 1 = i$ is shown explicitly to indicate that no $Q^{\varnothing}$ element is accessed. Additionally, to ensure connected states, only positions with d and $d+1$ on the same strand are included.

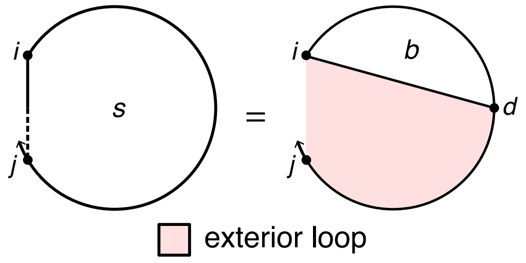


$$R^{\varnothing}_{\text{Multi}}(i,j,\phi) \equiv Q^{s}_{i,j} \oplus \bigoplus_{L,R \in \text{VP}(i,\max(j-4,n),\phi)} \text{dot}(\{Q^{\varnothing}_{i,L}, Q^{s}_{R,j}\}) \tag{A.23}$$

$$\text{where } n = \text{Last}(\eta)$$

Figure A.7: Polymer graph recursion diagram and recursion equation for the interstrand $R^{\varnothing}$ recursion.

Next, elements of the type $Q^{s}_{i,j}$ are computed using the recursion $R^{s}_{\text{Multi}}(i,j,\phi)$, shown in Figure A.8 and Equation A.24. This recursion collects contributions from all top level base pairs starting at i and ending in $[i+1,j]$. The contribution from each base pair $i \cdot d$ is incorporated through $Q^{b}_{i,d}$. No additional energy contribution is added as all these states are in exterior loops with no dangle or coaxial stacking. The index d must always be on the last strand to ensure there are no strand breaks in the subsequence $[d,j]$, which would lead to a disconnected state. The limiting case where $d=j$ is not depicted separately.

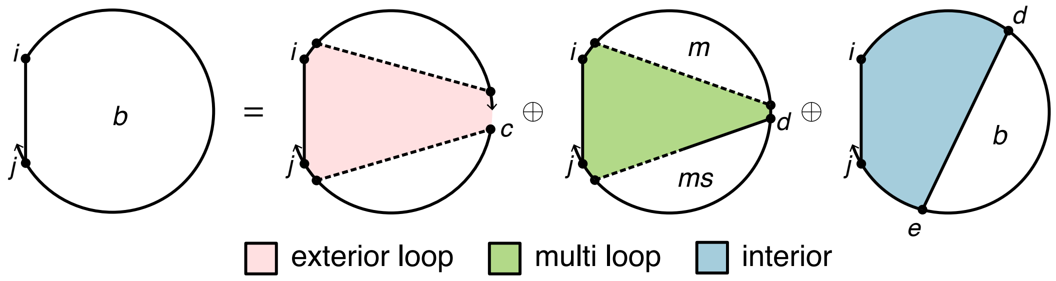


$$R^{s}_{\text{Multi}}(i,j,\phi) \equiv Q^{b}_{i,j} \oplus \begin{cases} \bigoplus_{d=n}^{j-1} Q^{b}_{i,d}, & n < j \\ \mathbb{0}, & \text{otherwise} \end{cases} \tag{A.24}$$

$$\text{where } n = \text{Last}(\eta)$$

Figure A.8: Polymer graph recursion diagram and recursion equation for the interstrand R^{s} recursion.

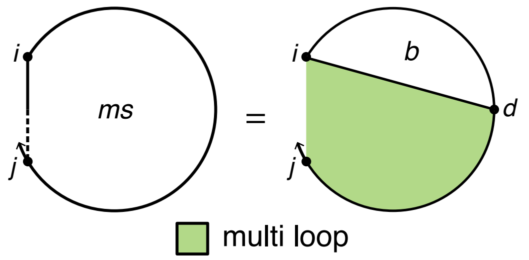
Elements of the type $Q^b_{i,j}$ are computed using the recursion $R^b_{\textsc{Multi}}(i, j, \phi)$, shown in Figure A.9 and Equation A.25. In the partition function calculation, this recursion computes the conditional partition function for subsequence $[i, j]$ given that i and j are base paired. In each contribution to $Q^b_{i,j}$, base pair $i \cdot j$ bounds one of three types of loop: a multiloop, and interior loop, or an exterior loop. Notably, there is no hairpin loop contribution in the interstrand recursion. If $i \cdot j$ were the only base pair in $[i, j]$ (the equivalent condition for a hairpin in the intrastrand recursion), then because there is at least one strand break in $[i, j]$, this would result in either an exterior loop (discussed below) or a disconnected structure, which is disallowed. In the multiloop cases, there are at least two other base pairs bounding the remainder of the multiloop. We recursively and uniquely consider all multiloop cases by obtaining the contribution from all $3'$−most base pairs starting at d and ending in $[d + 1, j]$ from $Q^{ms}_{d,j-1}$ and all possible other states of one or more base pairs in $[i + 1, d - 1]$ from $Q^m_{i+1,d-1}$. At this level of recursion, we account for the multiloop formation penalty, $\Delta G^{\text{multi}}_{\text{init}}$, as well as the branch penalty for base pair $i \cdot j$, $\Delta G^{\text{multi}}_{\text{bp}}$. To ensure all of these states are multiloops, there must never be a strand break between $d - 1$ and d. For the interior loop case, we defer discussion until Section A.3.3 where the two alternatives $\textsc{Multi}\ O(N^4)\ \textsc{Interior}(i, j, \phi)$ and $\textsc{Multi}\ O(N^3)\ \textsc{Interior}(i, j, \phi)$ are discussed. Finally, exterior loop cases are added by considering all positions c such that a strand break exists between $c - 1$ and c and incorporating the remaining sequence through $Q^{\varnothing}_{i+1,c-1}$ and $Q^{\varnothing}_{c,j-1}$. There is no explicit energy added for the exterior loop because coaxial and dangle stacking are not considered in this recursion.



$$R^b_{\text{MULTI}}(i,j,\phi) \equiv \begin{cases} C_1, & \text{PAIRABLE}(\phi_i,\phi_j) \\ \mathbb{0}, & \text{otherwise} \end{cases}$$
$$\text{where } m = \text{FIRST}(\eta)$$
$$n = \text{LAST}(\eta)$$
$$C_1 \equiv \begin{cases} \bigoplus_{c\in\eta} Q^{\varnothing}_{i+1,c-1} \otimes Q^{\varnothing}_{c,j-1}, & j \neq n \text{ and } i+1 \neq m \\ Q^{\varnothing}_{m,j-1}, & j \neq n \text{ and } i+1 = m \\ Q^{\varnothing}_{i+1,n-1}, & j = n \text{ and } i+1 \neq m \\ \mathbb{1}, & i+1 = j \end{cases} \tag{A.25}$$
$$\oplus \bigoplus_{L,R\in\text{VP}(i+1,j-1,\phi)} \text{DOT}(\{Q^m_{i+1,L}, Q^{ms}_{R,j-1}\}) \otimes W(\Delta G^{\text{multi}}_{\text{init}} + \Delta G^{\text{multi}}_{\text{bp}})$$
$$\oplus\ \text{MULTI INTERIOR}(i,j,\phi)$$

Figure A.9: Polymer graph recursion diagram and recursion equation for the interstrand R^b recursion.

Analogous to the recursion $R^s_{\text{MULTI}}(i,j,\phi)$, $R^{ms}_{\text{MULTI}}(i,j,\phi)$, shown in Figure A.10 and Equation A.26, collects contributions from all base pairs starting at i and ending in $[i+1,j]$. However, in this recursion, each base pair and set of unpaired bases are in a multiloop context. Thus, we obtain the contribution of the base-paired region from $Q^b_{i,d}$. This is combined with the branch penalty for base pair $i \cdot d$, $\Delta G^{\text{multi}}_{\text{bp}}$, as well as the contribution from the unpaired bases, $\Delta G^{\text{multi}}_{\text{nt}}(j-d)$. Note that in the dot product the range multiplying $\Delta G^{\text{multi}}_{\text{nt}}$ runs in reverse order because $j-d$ decreases in size as $d-i$ increases in size. The index d must always be on the last strand to ensure there are no strand breaks in the subsequence $[d,j]$, which would lead to either a disconnected or exterior loop state, which this recursion does not handle.

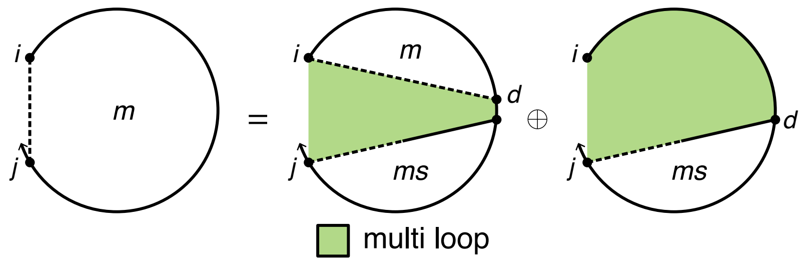


$$R^{ms}_{\textsc{Multi}}(i,j,\phi) \equiv Q^{b}_{i,j} \otimes W(\Delta G^{\text{multi}}_{\text{bp}})$$
$$\oplus \begin{cases} \textsc{dot}(\{Q^{b}_{i,[n:j-1]}, W(\Delta G^{\text{multi}}_{\text{bp}} + \Delta G^{\text{multi}}_{\text{nt}}[1:j-n]^{r})\}), & n < j \\ \mathbb{0}, & \text{otherwise} \end{cases} \tag{A.26}$$

where $n = \textsc{Last}(\eta)$

Figure A.10: Polymer graph recursion diagram and recursion equation for the interstrand R^{ms} recursion.

To finish the discussion of multiloop contributions, we turn to recursion $R^{m}_{\textsc{Multi}}(i,j,\phi)$, shown in Figure A.11 and Equation A.27. There are two types of contributions: those with exactly one base pair and those with more than one base pair. The single base pair case is handled by collecting all states with base pairs starting at d and ending in $[d+1,j]$ through $Q^{ms}_{d,j}$. Relative to the $R^{m}_{\textsc{Single}}(i,j,\phi)$, the additional requirement that d be on the first strand prevents this recursion from including exterior loop or disconnected states. As the $Q^{ms}_{d,j}$ recursion contains the penalty for the base pairs it wraps, only the penalty for the unpaired bases is considered here: $\Delta G^{\text{multi}}_{\text{nt}}(d-i)$. The multiple base pair case is handled by obtaining information about the $3'$−most base pair through $Q^{ms}_{d+1,j}$ and a recursive reference to a smaller subsequence value of $Q^{m}_{i,d}$. The indices d and $d+1$ must be on the same strand to prevent including exterior loop states, which this recursion does not handle. In this case, the subsidiary recursions have handled all explicit multiloop energies.



$$R^m_{\text{Multi}}(i,j,\phi) \equiv Q^{ms}_{i,j} \oplus \bigoplus_{L,R\in \text{VP}(i,j,\phi)} \text{dot}(\{Q^m_{i,L}, Q^{ms}_{R,j}\})$$
$$\oplus \begin{cases} \text{dot}(\{Q^{ms}_{[i+1:m-1],j}, W(\Delta G^{\text{multi}}_{\text{nt}}[1:m-i-1])\}), & i+1<m \\ \mathbb{0}, & \text{otherwise} \end{cases} \tag{A.27}$$

where $m = \text{First}(\eta)$

Figure A.11: Polymer graph recursion diagram and recursion equation for the interstrand R^m recursion.

A.3.3 Recursion contributions for interior loop contributions

Interior loop contributions to the recursions $R^b_{\text{Single}}(i,j,\phi)$ and $R^b_{\text{Multi}}(i,j,\phi)$ run naively in $O(N^4)$ time. This can be seen in the interior loop recursion diagrams in Figures A.4 and A.9, in which there are two free indices, d and e. This leads to $O(N^2)$ interior loop contributions per each of $O(N^2)$ base pairs $i\cdot j$, for an overall $O(N^4)$ complexity.

The intrastrand $O(N^4)$ interior loop contribution (Equation A.28) considers interior loops through a nested iteration, first over d in a 5′ to 3′ direction and for each d over e in a 5′ to 3′ direction. The index limits for d and e ensure $d-e \geq 4$, which corresponds to the minimum hairpin size of 3 unpaired nucleotides. The function $\Delta G^{\text{interior}}_{i,d,e,j}(\phi)$ accounts for the free energy of the loop with bouding base pairs $i\cdot j$ and $d\cdot e$. Depending on the indices, it substitutes in the correct energy form for a stack loop, bulge loop, or interior loop.

$$\text{Single } O(N^4) \text{ Interior}(i,j,\phi) \equiv \bigoplus_{d=i+1}^{j-5} \bigoplus_{e=d+4}^{j-1} \{Q^b_{d,e} \otimes W(\Delta G^{\text{interior}}_{i,d,e,j}(\phi))\} \tag{A.28}$$

The interstrand $O(N^4)$ interior loop contribution (Equation A.29) proceeds in the same general manner, considering interior loops in order of ascending d then e indices. However, d is restricted to be on the first strand and e is restricted to be on the last strand. This is reflected in the upper limit for d and the lower limit for e. These two requirements ensure that no strand breaks occur between i and d and between e and j, which would lead to an exterior loop or disconnected structure.

$$\textsc{Multi } O(N^4) \textsc{ Interior}(i, j, \phi) \equiv \bigoplus_{d=i+1}^{m-1} \bigoplus_{e=n}^{j-1} \{Q^b_{d,e} \otimes W(\Delta G^{\text{interior}}_{i,d,e,j}(\phi))\} \tag{A.29}$$

$$\begin{aligned} \text{where } m &= \textsc{First}(\eta) \\ n &= \textsc{Last}(\eta) \end{aligned}$$

To reduce the complexity of computing all interior loop contributions from $O(N^4)$ to $O(N^3)$, we take advantage of the specific energy form for large interior loops. In a previous discussion[14], this optimization was referred to as the "fastiloops" or "fast interior loops" function. Interior loops, defined by two bounding base pairs $i \cdot j$ and $d \cdot e$, can be classified by the distances $L_1 = d - i - 1$ and $L_2 = j - e - 1$. In cases where $L_1 < 4$ or $L_2 < 4$, the energy functions generally depend on terms that are non-linear with respect to L_1 and L_2. These include the special case energy functions for stack loops and bulge loops, as well as length dependent asymmetry and size penalties for other interior loops. We term these the *inextensible* interior loops because there is not a general purpose way to reuse previously computed information. There are only $O(N)$ of these contributions per subsequence (because of the constant upper bounds for L_1 and L_2), and so they do not contribute to the $O(N^4)$ complexity.

The remaining interior loops in which $L_1 \geq 4$ and $L_2 \geq 4$ are referred to as *extensible* interior loops. The number of such cases scales as $O(N^2)$: these are the cases that we must deal with efficiently to achieve an overall $O(N^3)$ runtime. For these cases, though, the free energy reduces to the following equation:

$$\Delta G^{\text{interior}}_{i,d,e,j}(\phi) = \gamma^{\text{size}}_{L_1+L_2} + \gamma^{\text{asymm}}_{|L_1-L_2|} + \gamma^{\text{mm}}_{j-1,j,i,i+1}(\phi) + \gamma^{\text{mm}}_{d-1,d,e,e+1}(\phi). \tag{A.30}$$

Here, the quantity $\gamma^{\text{size}}_{L_1+L_2}$ is a sequence-independent free energy contribution due to the size of interior loop, $L_1 + L_2$. The quantity $\gamma^{\text{asymm}}_{|L_1-L_2|}$ is a sequence-independent free energy contribution due to the *asymmetry* of the loop, the difference in the two side lengths, $|L_1 - L_2|$. Finally, the two terms $\gamma^{\text{mm}}_{j-1,j,i,i+1}(\phi)$ and $\gamma^{\text{mm}}_{d-1,d,e,e+1}(\phi)$ are sequence-dependent free energy contributions due to mismatch stacking on the base pairs $i \cdot j$ and $d \cdot e$, respectively.

Two key insights from Equation A.30 allow us to use this functional form to reduce complexity. First, for every base pair $i \cdot j$, the mismatch term for that base pair is independent of the other quantities and can be factored out. Second, every extensible loop bound by $i \cdot j$ can be converted to an extensible loop bound by $i - 1 \cdot j + 1$ by updating γ^{size}_s to $\gamma^{\text{size}}_{s+2}$ and replacing $\gamma^{\text{mm}}_{j-1,j,i,i+1}(\phi)$ with $\gamma^{\text{mm}}_{j,j+1,i-1,i}(\phi)$. Thus, we can cache the information specific to the base pair $d \cdot e$ for each given asymmetry the first time it is encountered in an extensible interior loop and then modify only the size information each time it is encountered.

Equation A.31 combines the above ideas into a subroutine for computing the interior loop contributions to $R^b_{\textsc{Single}}(i, j, \phi)$. The first three lines handle the inextensible interior loop cases. In order, these are cases where both $L_1 < 4$ and $L_2 < 4$, cases where $L_1 < 4$ and $L_2 \geq 4$, and cases where $L_1 \geq 4$ and $L_2 < 4$. They all use previously computed Q^b values and account for the interior loop free energy using the same function $\Delta G^{\text{interior}}_{i,d,e,j}(\phi)$ as in the $O(N^4)$ intrastrand recursion. Then, the extensible cases are handled by combining the already computed $Q^x_{i,j,s}$ for each loop size s with the mismatch contribution of base pair $i \cdot j$. The upper limit on s is set such that the minimum distance between d and e is 4, which corresponds to the minimum hairpin size of 3 unpaired nucleotides.

$$\begin{aligned}
\textsc{Single } O(N^3) \textsc{ Interior}(i, j, \phi) \equiv & \bigoplus_{d=i+1}^{i+4} \bigoplus_{e=j-4}^{j-1} Q^b_{d,e} \otimes W(\Delta G^{\text{interior}}_{i,d,e,j}(\phi)) \\
& \oplus \bigoplus_{d=i+1}^{i+4} \bigoplus_{e=d+4}^{j-5} Q^b_{d,e} \otimes W(\Delta G^{\text{interior}}_{i,d,e,j}(\phi)) \\
& \oplus \bigoplus_{e=j-4}^{j-1} \bigoplus_{d=i+5}^{e-4} Q^b_{d,e} \otimes W(\Delta G^{\text{interior}}_{i,d,e,j}(\phi)) \qquad \text{(A.31)} \\
& \oplus \textsc{Single Extensible}(i, j, \phi)
\end{aligned}$$

$$\text{where } \textsc{Single Extensible}(i, j, \phi) \equiv \bigoplus_{s=8}^{j-i-6} Q^x_{i,j,s} \otimes W(\gamma^{\text{mm}}_{j-1,j,i,i+1}(\phi))$$

The recursion $R^x_{\textsc{Single}}(i, j, s, \phi)$ (Equation A.32) fills in the three dimensional tensor Q^x referred to in Equation A.31. The indices i and j refer to the closing base pair $i \cdot j$ while the index s refers to the size of the extensible loops collected in $Q^x_{i,j,s}$. The contributions can be divided into two classes: previously encountered loops and new loops. The previously encountered loops are incorporated by accessing the previously computed element $Q^x_{i+1,j-1,s-2}$ and replacing $\gamma^{\text{size}}_{s-2}$ with γ^{size}_{s}. This is the key operation that reduces the complexity of the interior loop recursion to $O(N)$ by capturing all previous loops in $O(1)$. This component of the recursion is shown as the first term in the sum that defines C_1. All of the remaining contributions are from the new extensible loops that are first encountered for the indices i, j, s. These are the elements that have exactly $L_1 = 4$ or $L_2 = 4$ (or both).

Note that for each diagonal length $l = j - i + 1$, only elements of the form $Q^x_{i+1,j-1,s-2}$ are accessed for $O(N)$ values of s. Therefore, we only need to store elements of Q^x for diagonals $l, l-1$, and $l-2$. In other words, only Q^x values corresponding to Q^b values from the current and last 2 diagonals need to exist in memory during the forward pass. In moving to the next diagonal $l+1$, we can simply delete all $Q^x_{i,j,s}$ values for diagonal $l - 2$ as they will not be accessed again. As the number of Q^b elements per diagonal scales as $O(N)$ and the number of Q^x elements needed for each Q^b element scales as $O(N)$, only $O(N^2)$ space is necessary to store the necessary elements of Q^x. Naively storing all of $Q^x_{i,j,s}$ scales as $O(N^3)$, which would increase the space complexity of the presented algorithms from $O(N^2)$.

$$R^x_{\textsc{Single}}(i,j,s,\phi) \equiv \begin{cases} C_1, & j-i > 15 \text{ and } 10 \le s \le j-i-6 \\ C_2 + C_3, & j-i > 14 \text{ and } s = 9 \\ C_3, & j-i = 14 \text{ and } s = 9 \\ C_4, & j-i > 13 \text{ and } s = 8 \\ \mathbb{0}, & \text{otherwise} \end{cases}$$

$$\begin{aligned} \text{where } C_1 &\equiv W(\gamma^{\text{size}}_s - \gamma^{\text{size}}_{s-2}) \otimes Q^x_{i+1,j-1,s-2} \\ &\oplus W(\gamma^{\text{size}}_s + \gamma^{\text{asymm}}_{s-8} + \gamma^{\text{mm}}_{i+4,i+5,j+3-s,j+4-s}(\phi)) \otimes Q^b_{i+5,j+3-s} \\ &\oplus W(\gamma^{\text{size}}_s + \gamma^{\text{asymm}}_{s-8} + \gamma^{\text{mm}}_{s+i-4,s+i-3,j-5,j-4}(\phi)) \otimes Q^b_{s+i-3,j-5} \\ C_2 &\equiv W(\gamma^{\text{size}}_s + \gamma^{\text{asymm}}_{s-8} + \gamma^{\text{mm}}_{i+4,i+5,j-6,j-5}(\phi)) \otimes Q^b_{i+5,j-6} \\ C_3 &\equiv W(\gamma^{\text{size}}_s + \gamma^{\text{asymm}}_{s-8} + \gamma^{\text{mm}}_{i+5,i+6,j-5,j-4}(\phi)) \otimes Q^b_{i+6,j-5} \\ C_4 &\equiv W(\gamma^{\text{size}}_s + \gamma^{\text{asymm}}_{s-8} + \gamma^{\text{mm}}_{i+4,i+5,j-5,j-4}(\phi)) \otimes Q^b_{i+5,j-5} \end{aligned} \tag{A.32}$$

The subroutine for computing all interior loop contributions in $O(N^3)$ time for interstrand blocks is shown in Equation A.33. It follows an analogous structure to Equation A.31. The only differences are in the iteration limits which are all modified such that the invariants "d is on the same strand as i" and "e is on the same strand as j" are obeyed. Disobeying at least one of these invariants results in either an exterior loop or a disconnected structure.

$$\begin{aligned} \textsc{Multi } O(N^3) \textsc{ Interior}(i,j,\phi) \equiv & \bigoplus_{d=i+1}^{\min(i+4,m-1)} \bigoplus_{e=\max(n,j-4)}^{j-1} Q^b_{d,e} \otimes W(\Delta G^{\text{interior}}_{i,d,e,j}(\phi)) \\ \oplus & \bigoplus_{d=i+1}^{\min(i+4,m-1)} \bigoplus_{e=n}^{j-5} Q^b_{d,e} \otimes W(\Delta G^{\text{interior}}_{i,d,e,j}(\phi)) \\ \oplus & \bigoplus_{d=i+5}^{m-1} \bigoplus_{e=\max(n,j-4)}^{j-1} Q^b_{d,e} \otimes W(\Delta G^{\text{interior}}_{i,d,e,j}(\phi)) \\ \oplus & \textsc{ Multi Extensible}(i,j,\phi) \end{aligned} \tag{A.33}$$

$$\begin{aligned} \text{where } m &= \textsc{First}(\eta) \\ n &= \textsc{Last}(\eta) \end{aligned}$$

$$\textsc{Multi Extensible}(i,j,\phi) \equiv \bigoplus_{s=8}^{j-n+m-i-2} Q^x_{i,j,s} \otimes W(\gamma^{\text{mm}}_{j-1,j,i,i+1}(\phi))$$

The recursion $R^x_{\textsc{Multi}}(i,j,s,\phi)$ (Equation A.34) is also a slight modification of its single stranded counterpart. The recursive component that extends previously encountered extensible loops is shown in C_5. The remainder of contributions are the newly encountered extensible loops. The increase in the number of cases results from the checks necessary to ensure that no exterior loops are spuriously included.

$$
R^x_{\textsc{Multi}}(i,j,s,\phi) \equiv \begin{cases} C_1, & i+6<m \text{ and } n+6 \le j \\ C_2, & i+6=m \text{ and } n+6 \le j \text{ and } 9 \le s < j+4-n \\ C_3, & i+6<m \text{ and } n+5=j \text{ and } 9 \le s < m-i+3 \\ C_4, & i+5<m \text{ and } n+5 \le j \text{ and } s=8 \\ \mathbb{0}, & \text{otherwise} \end{cases}
$$

$$
\begin{aligned}
\text{where } m &= \textsc{First}(\eta) \\
n &= \textsc{Last}(\eta) \\
r &= j+4-n \\
t &= m-i+3 \\
p &= j-n+m-i-2 \\
C_1 &\equiv \begin{cases} C_5+C_6+C_7, & 10 \le s < \min(r,t) \\ C_5, & \max(r,t) < s \le p \\ C_6, & r < s \le t \\ C_7, & t < s \le r \\ C_8, & s=9 \\ \mathbb{0}, & \text{otherwise} \end{cases} \\
C_2 &\equiv W(\gamma^{\text{size}}_s + \gamma^{\text{asymm}}_{s-8} + \gamma^{\text{mm}}_{i+4,i+5,j+3-s,j+4-s}(\phi)) \otimes Q^b_{i+5,j+3-s} \\
C_3 &\equiv W(\gamma^{\text{size}}_s + \gamma^{\text{asymm}}_{s-8} + \gamma^{\text{mm}}_{s+i-4,s+i-3,j-5,j-4}(\phi)) \otimes Q^b_{s+i-3,j-5} \\
C_4 &\equiv W(\gamma^{\text{size}}_s + \gamma^{\text{asymm}}_{s-8} + \gamma^{\text{mm}}_{i+4,i+5,j-5,j-4}(\phi)) \otimes Q^b_{i+5,j-5} \\
C_5 &\equiv W(\gamma^{\text{size}}_s - \gamma^{\text{size}}_{s-2}) \otimes Q^x_{i+1,j-1,s-2} \\
C_6 &\equiv W(\gamma^{\text{size}}_s + \gamma^{\text{asymm}}_{s-8} + \gamma^{\text{mm}}_{i+4,i+5,j+3-s,j+4-s}(\phi)) \otimes Q^b_{i+5,j+3-s} \\
C_7 &\equiv W(\gamma^{\text{size}}_s + \gamma^{\text{asymm}}_{s-8} + \gamma^{\text{mm}}_{s+i-4,s+i-3,j-5,j-4}(\phi)) \otimes Q^b_{s+i-3,j-5} \\
C_8 &\equiv W(\gamma^{\text{size}}_s + \gamma^{\text{asymm}}_{s-8} + \gamma^{\text{mm}}_{i+5,i+6,j-5,j-4}(\phi)) \otimes Q^b_{i+6,j-5} \\
&\quad \oplus W(\gamma^{\text{size}}_s + \gamma^{\text{asymm}}_{s-8} + \gamma^{\text{mm}}_{i+4,i+5,j-6,j-5}(\phi)) \otimes Q^b_{i+5,j-6}
\end{aligned}
\tag{A.34}
$$

Given the preceding discussion of the work necessary to reduce the time complexity of the interior loop recursions to $O(N^3)$ while maintaining $O(N^2)$ storage, is there any use case for the $O(N^4)$ recursions? During algorithms using a forward sweep operation order, the only useful case is as an internal consistency check for the $O(N^3)$ recursions. However, the backtracking algorithms do require the $O(N^4)$ interior loop recursions. Unlike the forward sweep in which all recursion types for all i and j are evaluated, backtracking generally requires reevaluation of only a subset of all possible recursions. This is incompatible with the optimization of throwing away Q^x values that are no longer necessary during the forward pass. However, as discussed in Section A.6.5, this does not impact the worst-case time complexity for backtracking.

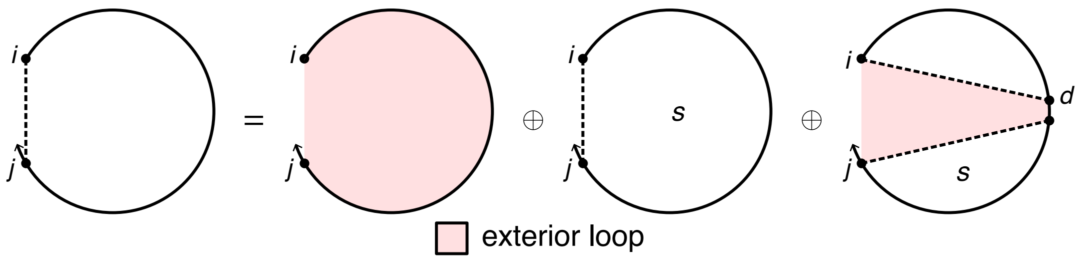
A.4 Recursions including coaxial stacking and dangles

Deriving recursions for the the full secondary structure model including coaxial stacking and dangle stacking resulted in several changes to the recursions in Section A.3. In general, the naming, renaming, and introduction of new recursions was balanced to limit proliferation of new names and reduce confusion with the non-stacking, non-dangling recursions. Some existing recursions, such as R^b and $R^\varnothing$, that can be thought of as conditional partition functions of a succintly described structural ensemble continue to have this same meaning. In the case of $R^\varnothing$, the form of the recursion is precisely the same. For R^b, the contributions defining the recursion were changed to handle contributions coming from coaxial stacking in multiloops. Nevertheless, R^b still collects all contributions from states where the endpoints are paired. The recursion R^m functions as a collector for "the rest of" a multiloop state, but is now defined in terms of recursions that account for multiloop coaxial and dangle stacking. The recursion R^s retains the same name because it acts in a conceptually similar way. Specifically, it reduces a naively $O(N^4)$ set of contributions to $O(N^3)$ by collecting all contributions with a fixed $5'$−most nucleotide. In this set of recursions, it wraps contributions from R^c analogously to it wrapping contributions from R^b in the non-stacking, non-dangle recursions.

Additional recursion types are added to deal specifically with contributions coming from dangle and coaxial stacking states. All of these recursions have identifiers starting with "c" (R^c, R^{c_s}, R^{c_m}, $R^{c_s s}$, and $R^{c_b s}$), which can be thought of as standing for "coaxial". The recursion R^c in the exterior loop context is analogous to the sum of the two recursions R^{c_m} and R^{c_s} in the multiloop context. The dangle (R^{c_m}) and coaxial (R^{c_s}) stacking contributions in multiloop contexts are computed separately, so that an edge case can be dealt with correctly. The recursion $R^{c_b s}$ functions analogously to R^s, but in a multiloop context instead of an exterior loop one. Finally, the recursion $R^{c_s s}$ wraps only the coaxial stacking contributions in a multiloop context (for the same edge case mentioned above). We generically refer to states that involve either a dangle or coaxial stacking state as *stacking states*.

A.4.1 Intrastrand dynamic programming recursions

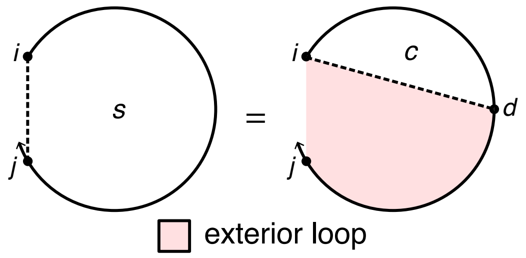
We begin with the recursion $R^\varnothing_{\textsc{Single}}(i, j, \phi)$ shown in Figure A.12 and Equation A.35. In the partition function calculation, this recursion computes the unconstrained partition function for subsequence $[i, j]$. This recursion distinguishes two cases: those that have at least one top level stacking state (i.e. a stacking state in an exterior loop context) and the empty case where there are no stacking states in $[i, j]$. The empty case is $\mathbb{1}$ because a secondary structure with no base pairs is defined to be the reference state with free energy 0. In cases with at least one top level stacking state, there is a $3'$−most stacking state which begins at $d + 1$ and ends in the interval $[d + 2, j]$ and is incorporated through $Q^s_{d+1,j}$. The subsequence $[i, d]$ is incorporated through recursive access of the dependent value $Q^\varnothing_{i,d}$. The limiting case where the index $d + 1 = i$ is shown explicitly to indicate that no $Q^\varnothing$ element is accessed.



$$R^{\varnothing}_{\textsc{Single}}(i,j,\phi) \equiv \mathbb{1} \oplus \begin{cases} Q^s_{i,j} \oplus \textsc{dot}(\{Q^{\varnothing}_{i,[i:j-5]}, Q^s_{[i+1:j-4],j}\}), & j-i>4 \\ Q^s_{i,j}, & j-i=4 \\ \mathbb{0}, & \text{otherwise} \end{cases} \tag{A.35}$$

Figure A.12: Polymer graph recursion diagram and recursion equation for the intrastrand R recursion.

Following the recursion to the only subsidiary recursion type accessed in $R^{\varnothing}$, elements of the type $Q^s_{i,j}$ are computed using the recursion $R^s_{\textsc{Single}}(i,j,\phi)$, shown in Figure A.13 and Equation A.36. This recursion collects contributions from all top level stacking states starting at i and ending in $[i+1,j]$. The contribution from each stacking state over subsequence $\phi_{i,d}$ is incorporated through $Q^c_{i,d}$. This generalized "summing" is the reason for the s name. No additional energy contribution is added as it was already handled during R^c. The explicit lower index bound of the non-dangling, non-stacking recursions is omitted as the dependent recursions will return $\mathbb{0}$ for subsequences too small to have a stacking state.

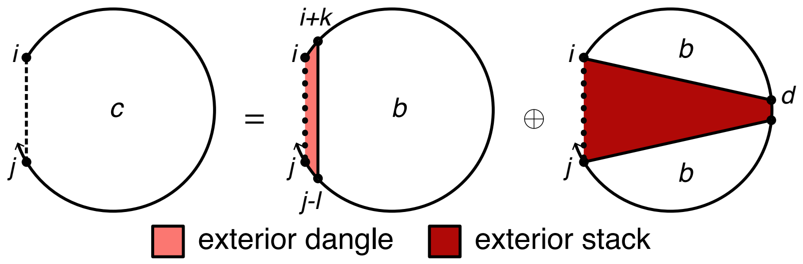


$$R^s_{\textsc{Single}}(i,j,\phi) \equiv \bigoplus_{d=i}^{j} Q^c_{i,d} \tag{A.36}$$

Figure A.13: Polymer graph recursion diagram and recursion equation for the intrastrand R^s recursion.

The recursion R^s is a simple efficiency wrapper on $R^c_{\textsc{Single}}(i,j,\phi)$, shown in Figure A.14 and Equation A.37. This recursion captures dangle and stacking states in exterior loop contexts and accounts for the free energy contributions of these states. For dangle stacking states, four possible base pairs are considered of the form $i+k \cdot j-l$, where k and l can be either 0 or 1. When $k=0$

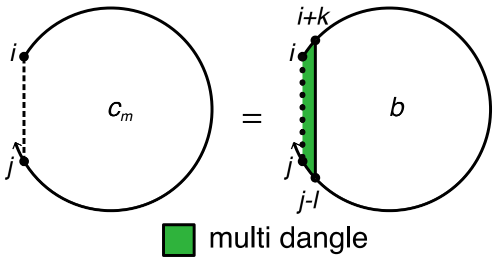
and $l = 0$, we have the no-dangle case for base pair $i \cdot j$. When $k = 1$ and $l = 0$, we have nucleotide i dangling on base pair $i+1 \cdot j$. When $k = 0$ and $l = 1$, we have nucleotide j dangling on base pair $i \cdot j-1$. When $k = 1$ and $l = 1$, we have nucleotides i and j simultaneously dangling on base pair $i+1 \cdot j-1$ in a *terminal mismatch* state. The details of the free energy contributions are abstracted by the function $\Delta G^{\text{dangle}}_{i,i+k,j-l,j}(\phi)$ which redirects to the correct free energy function based on context. These states are all summarized by the first contribution recursion diagram in Figure A.14. The second contribution recursion diagram summarizes how the coaxial stacking states are considered. Every pair of adjacent valid base pairs $i \cdot d$ and $d+1 \cdot j$ are added in states where they coaxially stack on each other. The free energy of this contribution is added through the function $\Delta G^{\text{coax}}_{i,d,j}(\phi)$, which only requires 3 indices as $d+1$ is implied by d. In all of these cases, the contribution of the subsequence(s) within the base-paired subsequence is incorporated through the Q^b element(s).



$$R^c_{\text{SINGLE}}(i,j,\phi) \equiv \bigoplus_{\substack{k\in\{0,1\}\\ l\in\{0,1\}}} \begin{cases} Q^b_{i+k,j-l} \otimes W(\Delta G^{\text{dangle}}_{i,i+k,j-l,j}(\phi)), & i+k < j-l \\ \mathbb{0}, & \text{otherwise} \end{cases}$$
$$\oplus \begin{cases} \textsc{dot}(\{Q^b_{i,[i:j-1]}, Q^b_{[i+1:j],j}, W(\Delta G^{\text{coax}}_{i,[i:j-1],j}(\phi))\}), & i < j \\ \mathbb{0}, & \text{otherwise} \end{cases} \tag{A.37}$$

Figure A.14: Polymer graph recursion diagram and recursion equation for the intrastrand R^c recursion.

These same two types of stacking states appear within multiloop contexts, albeit with additional multiloop-specific energy contributions. However, the requirement that multiloops have at least three bounding base pairs creates edge cases where coaxial stacking states need to be distinguishable from dangle stacking states when refered to by R^b. So, the multiloop equivalent of R^c is split into two separate recursions: R^{c_m}, which handles the dangle states, and R^{c_s}, which handles the coaxial stacking states. The recursion for $R^{c_m}_{\text{SINGLE}}(i,j,\phi)$ is shown in Figure A.15 and Equation A.38. The free energy contribution from the dangle state is handled in the same way as the analogous states included in R^c. The subscript m in the recursion name can be thought of as standing for "*multiloop* dangle" contributions (with d not being used to avoid confusion with its frequent use as an index). Because the dangling is occuring in a multiloop context, we also account for the multiloop branch penalty for base pair $i+k \cdot j-l$ with $\Delta G^{\text{multi}}_{\text{bp}}$. Additionally, for whichever nucleotides are dangling in each state, these unpaired nucleotides are penalized appropriately with an equal number of $\Delta G^{\text{multi}}_{\text{nt}}$ terms.

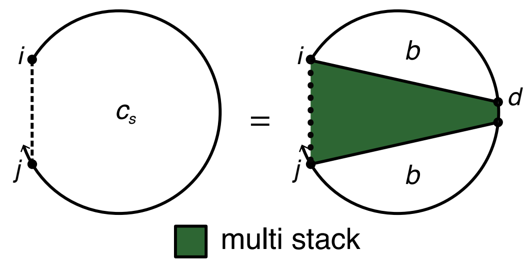


$$R^{c_m}_{\textsc{Single}}(i,j,\phi) \equiv \bigoplus_{\substack{k\in\{0,1\}\\ l\in\{0,1\}}} \begin{cases} Q^b_{i+k,j-l} \otimes W(\Delta G^{\text{dangle}}_{i,i+k,j-l,j}(\phi) + \Delta G^{\text{multi}}_{\text{bp}} + (k+l)\Delta G^{\text{multi}}_{\text{nt}}) & i+k < j-l \\ \mathbb{0}, & \text{otherwise} \end{cases} \tag{A.38}$$

Figure A.15: Polymer graph recursion diagram and recursion equation for the intrastrand R^{c_m} recursion.

The recursion $R^{c_s}_{\textsc{Single}}(i,j,\phi)$, shown in Figure A.16 and Equation A.39, handles the coaxial stacks in multiloops in an analogous way to R^c handling these states in exterior loops. The subscript s in the recursion name can be thought of as standing for "multiloop coaxial *stack*" contributions. The same function $\Delta G^{\text{coax}}_{i,d,j}(\phi)$ is used to look up the appropriate coaxial stack free energy to include. The only difference is that R^{c_s} accounts for the multiloop branch penalties for base pairs $i \cdot d$ and $d+1 \cdot j$ with two counts of $\Delta G^{\text{multi}}_{\text{bp}}$. As the base pairs $i \cdot d$ and $d+1 \cdot j$ perfectly bisect the sequence $[i,j]$ into Q^b elements, no $\Delta G^{\text{multi}}_{\text{nt}}$ terms are needed to account for unpaired bases.

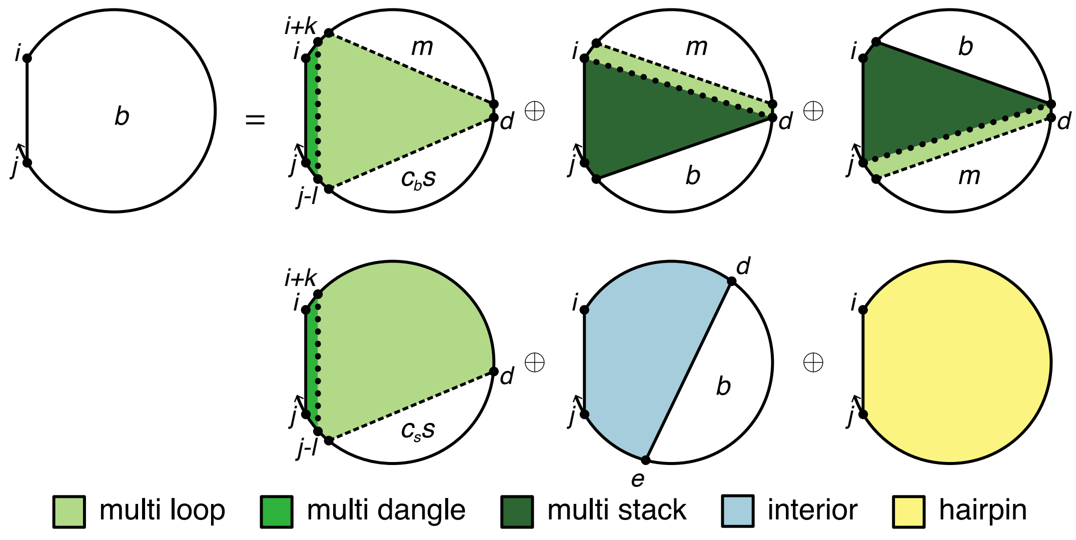
Finally, there is one point to note about the appearance of the recursion diagram for R^{c_s}. We are not perfectly consistent in our use of the dashed line because we know for every state captured in R^{c_s}, both nucleotides i and j are in a base pair. If we were to use the half-solid, half-dashed convention, we would need two half-solid lines which is just a solid line. That would imply incorrectly that i and j are base paired. To avoid this confusion, we drop use of the half-solid, half-dashed convention for Figures A.16, A.18, A.25, and A.28 and allow the recursion definitions to indicate where base pairs may exist.



$$R^{c_s}_{\textsc{Single}}(i,j,\phi) \equiv \begin{cases} \textsc{dot}(\{Q^b_{i,[i:j-1]}, Q^b_{[i+1:j],j}, W(\Delta G^{\text{coax}}_{i,[i:j-1],j}(\phi))\}) \otimes W(2\Delta G^{\text{multi}}_{\text{bp}}), & i < j \\ \mathbb{0}, & \text{otherwise} \end{cases} \tag{A.39}$$

Figure A.16: Polymer graph recursion diagram and recursion equation for the intrastrand R^{c_s} recursion.

In the non-dangling, non-stacking recursions, R^b had the most complicated set of recursion contributions. This recursion is further complicated by the addition of multiloop coaxial and dangle stacking contributions, as shown for $R^b_{\textsc{Single}}(i,j,\phi)$ in Figure A.17 and Equation A.40. The recursion is unchanged for interior loop and hairpin loop contributions. For interior loops, it uses the same functions Single $O(N^4)$ Interior(i,j,ϕ) and Single $O(N^3)$ Interior(i,j,ϕ) discussed in Section A.3.3. The hairpin loop contribution is precisely the same as in Equation A.20. Multiloop contributions are complicated enough that they have been split off into their own subroutines, Single Closing Pair Stacking and Single Closing Pair Dangles.



$$\begin{aligned} R^b_{\textsc{Single}}(i,j,\phi) \equiv\; & \textsc{Single Closing Pair Stacking}(i,j,\phi) \\ & \oplus \textsc{Single Closing Pair Dangles}(i,j,\phi) \\ & \oplus \textsc{Single\ \ Interior}(i,j,\phi) \\ & \oplus W(G^{\text{hairpin}}(i,j,\phi)) \end{aligned} \tag{A.40}$$

Figure A.17: Polymer graph recursion diagram and recursion equation for the intrastrand R^b recursion.

The subroutine Single Closing Pair Stacking, shown in Equation A.41, handles the states where a base pair adjacent to $i \cdot j$ in $[i+1, j-1]$ (either $i+1$ or $j-1$ base paired) stacks on $i \cdot j$. These cases are depicted in Figure A.17 by the rightmost two recursion diagrams in the first row. These cases must be considered in $R^b_{\textsc{Single}}(i,j,\phi)$ separately from coaxial stacks occurring wholly in the sequence $[i+1, j-1]$ as this is the first time $i \cdot j$ is encountered. As the recursion collects the free energy of the coaxial stack between $i \cdot j$ and the adjacent base pair, the contribution from the sequence within the adjacent base pair is incorporated through the dependent $Q^b_{i+1,d-1}$ or $Q^b_{d,j-1}$ element and the remainder of the multiloop contributions through the $Q^m_{d,j-1}$ or $Q^m_{i+1,d-1}$ element, respectively. The explicit coaxial stack energy is calculated with the function calls $\Delta G^{\text{coax}}_{j,i,d-1}(\phi)$ or $\Delta G^{\text{coax}}_{d,j-1,i}(\phi)$, respectively. Additionally, as we are in a multiloop and explicitly considering one closing base pair and one other pair, we account for the multiloop formation penalty, $\Delta G^{\text{multi}}_{\text{init}}$, and a branch penalty, $\Delta G^{\text{multi}}_{\text{bp}}$, for each pair. Every nucleotide position is either base paired or handled by a previous recursion, and so no $\Delta G^{\text{multi}}_{\text{nt}}$ terms are needed to account for unpaired bases.

$$\textsc{Single Closing Pair Stacking}(i, j, \phi) \equiv \begin{cases} C_1, & j - i > 2 \\ \mathbb{0}, & \text{otherwise} \end{cases}$$

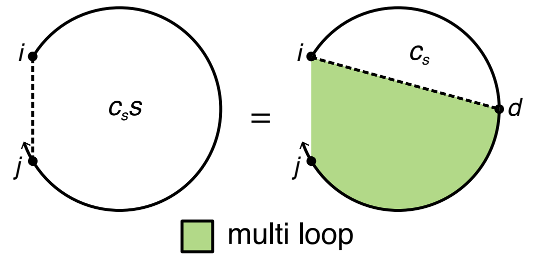
$$\begin{aligned} \text{where } C_1 \equiv\ & \textsc{dot}(\{Q^b_{i+1,[i+1:j-2]}, Q^m_{[i+2:j-1],j-1}, W(\Delta G^{\text{coax}}_{j,i,[i+1:j-2]}(\phi))\}) \otimes W(\Delta G^{\text{multi}}_{\text{init}} + 2\Delta G^{\text{multi}}_{\text{bp}}) \\ & \oplus \textsc{dot}(\{Q^m_{i+1,[i+1:j-2]}, Q^b_{[i+2:j-1],j-1}, W(\Delta G^{\text{coax}}_{[i+2:j-1],j-1,i}(\phi))\}) \otimes W(\Delta G^{\text{multi}}_{\text{init}} + 2\Delta G^{\text{multi}}_{\text{bp}}) \end{aligned} \tag{A.41}$$

The subroutine Single Closing Pair Dangles, shown in Equation A.42, handles the remaining multiloop states in which the base pair $i \cdot j$ does not coaxially stack with other other base pairs within the multiloop it bounds. Unpaired nucleotides in the multiloop may, however, dangle stack on base pair $i \cdot j$. Figure A.17 depicts these cases in the contribution recursion diagrams in the first column. The top diagram is roughly analogous to the way multiloop contributions are handled in Figure A.4. However, instead of splitting off all $3'$−most base pairs, this recursion splits off all $3'$−most multiloop stacking states with contribution $Q^{c_b s}_{d,j-l-1}$. The $R^{c_b s}$ recursion, discussed below, captures both single base pairs as well as coaxial stacking states (two base pairs) in a multiloop context. The remainder of the multiloop contribution is accounted for through the element $Q^m_{i+k+1,d-1}$. The bottom diagram depicts all states in an edge case where the are exactly three base pairs and the two subsequence base pairs coaxially stack on each other. This case is not captured in the cases in the top diagram. States in the top diagram with exactly three base pairs *cannot* impose coaxial stacking between the two inner pairs, as the two base pairs are incorporated independently through two separate prior recursions. This requires accumulating the explicitly coaxial stack information in $Q^{c_s s}$ and accessing it here instead of accessing $Q^{c_b s}$, which has both coaxial and dangle stacking information. In both sets of cases, dangle stacking on base pair $i \cdot j$ is handled in the same way discussed for the recursion R^c. The only difference for these states is that the base pair itself is fixed while k and l determine whether the adjacent nucleotides stack or not. Any dangling nucleotides and all of the explicitly unpaired nucleotides in the latter type of contribution are accounted for by the correct number of multiloop unpaired nucleotide penalties, $\Delta G^{\text{multi}}_{\text{nt}}$.

$$\textsc{Single Closing Pair Dangles}(i, j, \phi) \equiv \bigoplus_{\substack{k \in \{0,1\} \\ l \in \{0,1\}}} \begin{cases} C_1, & i + k + 1 = j - l - 1 \\ C_1 \oplus C_2, & i + k + 1 < j - l - 1 \\ \mathbb{0}, & \text{otherwise} \end{cases}$$

$$\begin{aligned} \text{where } \Delta G^{\text{multi}}_{\text{closing}} &\equiv \Delta G^{\text{multi}}_{\text{init}} + \Delta G^{\text{multi}}_{\text{bp}} \\ C_1 &\equiv \textsc{dot}(\{Q^{c_s s}_{[i+k+1:j-l-1],j-l-1}, W(\Delta G^{\text{multi}}_{\text{closing}} + \Delta G^{\text{multi}}_{\text{nt}}[k+1 : j-i+l-2])\}) \\ &\quad \otimes W(\Delta G^{\text{dangle}}_{j-l,j,i,i+k}(\phi)) \\ C_2 &\equiv \textsc{dot}(\{Q^{m}_{i+k+1,[i+k+1:j-l-2]}, Q^{c_b s}_{[i+k+2:j-l-1],j-l-1}\}) \\ &\quad \otimes W(\Delta G^{\text{dangle}}_{j-l,j,i,i+k}(\phi) + \Delta G^{\text{multi}}_{\text{closing}} + \Delta G^{\text{multi}}_{\text{nt}}(k+l)) \end{aligned} \tag{A.42}$$

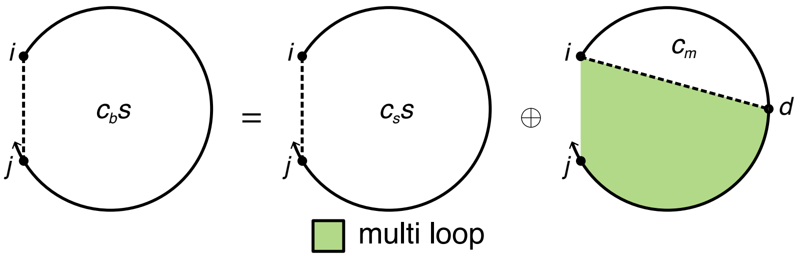
The multiloop contributions for the recursion R^b are dependent on three new recursions, $R^{c_s s}$, $R^{c_b s}$, and R^m, which have not yet been discussed. We describe them in the above order, which is from least to most dependent. The recursion $R^{c_s s}_{\textsc{Single}}(i, j, \phi)$, shown in Figure A.18 and Equation A.43, acts as an efficiency wrapper for elements of Q^{c_s}. It collects contributions from all multiloop coaxial stacking states starting at i and ending in $[i + 1, j]$. The contributions from the individual coaxial stacks are incorporated from $Q^{c_s}_{i,d}$. This is combined with the contribution from the unpaired bases, $\Delta G^{\text{multi}}_{\text{nt}}(j - d)$. No branch penalties are applied because they have been handled already in R^{c_s}.



$$R^{c_s s}_{\textsc{Single}}(i, j, \phi) \equiv \textsc{dot}(\{Q^{c_s}_{i,[i:j]}, W(\Delta G^{\text{multi}}_{\text{nt}}[0 : j - i]^r\}) \tag{A.43}$$

Figure A.18: Polymer graph recursion diagram and recursion equation for the intrastrand $R^{c_s s}$ recursion. The $R^{c_s s}$ recursion contains all contributions in which a multiloop coaxial stacking state begins at base i. There are contributions from the coaxial stacking states ending at each of the possible bases d in the subsequence, obtained from the R^{c_s} recursion.

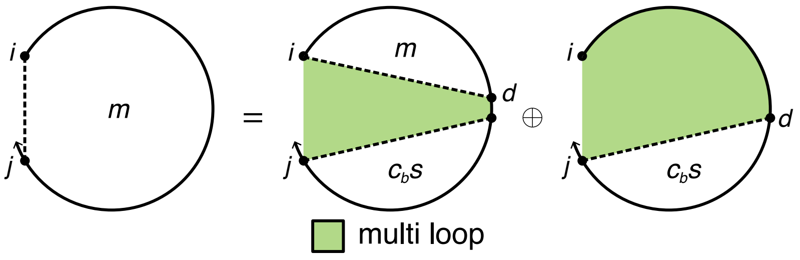
The recursion $R^{c_b s}_{\textsc{Single}}(i, j, \phi)$, shown in Figure A.19 and Equation A.44, acts as an efficiency wrapper for elements of both Q^{c_s} and Q^{c_m}. That is, it captures multiloop stacking states starting at i and ending in $[i + 1, j]$. The subscript b in the identifier for the recursion can be thought of as meaning "both" coaxial and dangle stacking states in a multiloop context. Because the contributions from the multiloop coaxial stacking states have already been accumulated in $Q^{c_s s}_{i,j}$, they are captured in this recursion through that element. Then the dangle states starting at i and ending in $[i + 1, j]$ are handled in an analogous way. The contributions from the individual dangle states are incorporated from $Q^{c_m}_{i,d}$. This is combined with the contribution from the unpaired bases, $\Delta G^{\text{multi}}_{\text{nt}}(j-d)$. No branch penalties are applied because they have already been handled in R^{c_m}.



$$R^{c_b s}_{\text{Single}}(i,j,\phi) \equiv Q^{c_s s}_{i,j} \oplus \text{dot}(\{Q^{c_m}_{i,[i:j]}, W(\Delta G^{\text{multi}}_{\text{nt}}[0:j-i]^r\}) \tag{A.44}$$

Figure A.19: Polymer graph recursion diagram and recursion equation for the intrastrand $R^{c_b s}$ recursion.

Finally the recursion $R^m_{\text{Single}}(i,j,\phi)$, shown in Figure A.20 and Equation A.45, captures the recursive "remainder" of the multiloop contribution for use in R^b. There are two cases for how this remainder can look: one additional multiloop stacking state or more than one of these multiloop stacking states. For multiple stacking states, the 3′−most stacking state is incorporated explicitly through the $Q^{c_b s}_{d+1,j}$ while the remainder is handled through a recursive reference to a smaller subsequence value of $Q^m_{i,d}$. No explicit multiloop, dangle, or coaxial stack free energies need be considered because they have been handled in the dependent elements. Otherwise, there is exactly one additional stacking state in the multiloop, which is incorporated here through $Q^{c_b s}_{d,j}$ along with multiloop unpaired nucelotide penalties for the explicitly unpaired nucleotides in $[i, d-1]$.



$$\begin{aligned} R^m_{\text{Single}}(i,j,\phi) \equiv\ & \text{dot}(\{Q^{c_b s}_{[i:j],j}\}, W(\Delta G^{\text{multi}}_{\text{nt}}[0:j-i]\}) \\ & \oplus \begin{cases} \text{dot}(\{Q^m_{i,[i:j-1]}, Q^{c_b s}_{[i+1:j],j}\}), & i<j \\ \mathbb{0}, & \text{otherwise} \end{cases} \end{aligned} \tag{A.45}$$

Figure A.20: Polymer graph recursion diagram and recursion equation for the intrastrand R^m recursion.

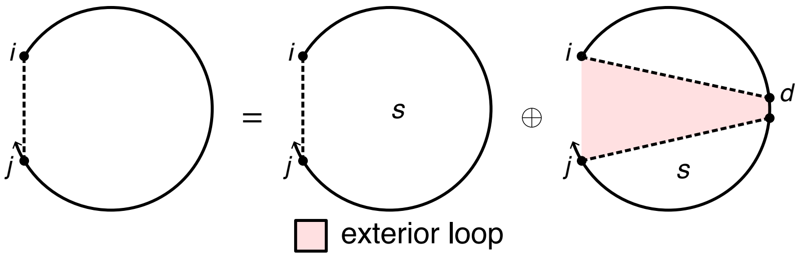
A.4.2 Interstrand dynamic programming recursions

The following subroutine, FS standing for *full stride*, is used in several of the multistranded recursions. It checks that index i is on the first strand and index j is on the last strand. Generally it is used to exclude states that would include an exterior loop incorrectly.

```
FS($i, j, \phi$)
    $b \leftarrow \eta$
    $m \leftarrow$ First($b$) // index of first base on second strand
    $n \leftarrow$ Last($b$) // index of first base on last strand
    **return** $i < m$ **and** $j \geq n$
```

Algorithm A.3: Procedure to determine if a pair of nucleotide indices are on the first and last strands in an interstrand block.

We begin with the recursion $R^{\varnothing}_{\text{Multi}}(i, j, \phi)$ shown in Figure A.21 and Equation A.46. In the partition function calculation, this recursion computes the unconstrained partition function for subsequence $[i, j]$. Unlike $R^{\varnothing}_{\text{Single}}(i, j, \phi)$, there is no empty case because this would correspond to a disconnected structure, which is not in the multistranded ensemble. Thus, there are only cases with one top level stacking state. These are collected by considering all $3'$−most stacking states beginning at $d + 1$ and ends in the interval $[d + 2, j]$ and is incorporated through $Q^{s}_{d+1,j}$. The subsequence $[i, d]$ is incorporated through recursive access of the dependent value $Q^{\varnothing}_{i,d}$. The limiting case where the index $d + 1 = i$ is shown explicitly to indicate that no $Q^{\varnothing}$ element is accessed.

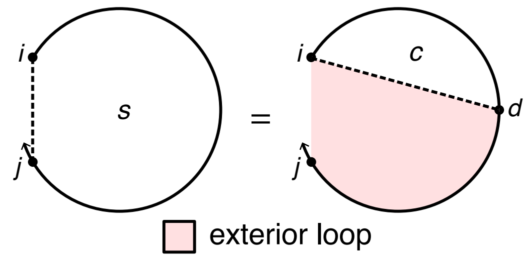


$$R^{\varnothing}_{\text{Multi}}(i, j, \phi) \equiv Q^{s}_{i,j} \oplus \bigoplus_{L,R \in \text{VP}(i, \max(j-4, n), \phi)} \text{dot}(\{Q^{\varnothing}_{i,L}, Q^{s}_{R,j}\}) \tag{A.46}$$

where $n = \text{Last}(\eta)$

Figure A.21: Polymer graph recursion diagram and recursion equation for the interstrand R recursion.

Following the recursion to the only type used in $R^{\varnothing}$, elements of the type $Q^s_{i,j}$ are computed using the recursion $R^s_{\textsc{Multi}}(i, j, \phi)$, shown in Figure A.22 and Equation A.47. This recursion collects contributions from all top level stacking states starting at i and ending in $[n, j]$. The lower bound on d ensures there are no strand breaks in the subsequence $[d, j]$, which would lead to a disconnected state. The contribution from each stacking state over subsequence $\phi_{i,d}$ is incorporated through $Q^c_{i,d}$. This generalized "summing" is the reason for the s name. No additional energy contribution is added as it was already handled during R^c.

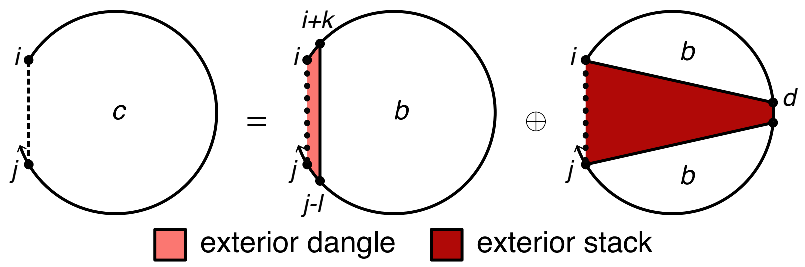


$$R^s_{\textsc{Multi}}(i, j, \phi) \equiv \bigoplus_{d=n}^{j} Q^c_{i,d} \tag{A.47}$$

$$\text{where } n = \textsc{Last}(\eta)$$

Figure A.22: Polymer graph recursion diagram and recursion equation for the interstrand R^s recursion.

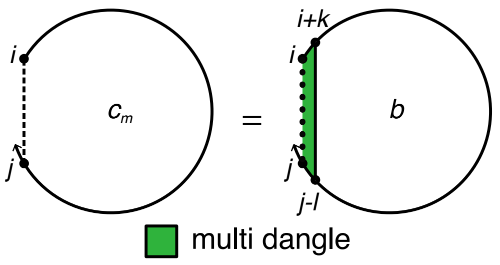
The recursion R^s wraps $R^c_{\textsc{Multi}}(i, j, \phi)$, shown in Figure A.23 and Equation A.48. This recursion captures dangle and stacking states in exterior loop contexts and accounts for the free energy contributions of these states. For dangle stacking states, four possible base pairs are considered of the form $i + k \cdot j - l$, where k and l can be either 0 or 1. These are the same states as in the $R^c_{\textsc{Single}}(i, j, \phi)$, and are handled analogously. In this multistraned setting, however, there is the additional requirement that the dangling nucleotides be on the same strand, as otherwise there would be a disconnected structure. These states are all summarized by the first contribution recursion diagram in Figure A.23. The second contribution recursion diagram summarizes how the coaxial stacking states are considered. The coaxial stacking state for every pair of adjacent valid base pairs $i \cdot d$ and $d + 1 \cdot j$ is accumulated. The free energy of this contribution is added through the function $\Delta G^{\text{coax}}_{i,d,j}(\phi)$, which only requires 3 indices as $d + 1$ is implied by d. These states are only valid if d and $d + 1$ are on the same strand, which is enforced by using the VP function to generate the possible indices. In all of these cases, the contribution of the subsequence(s) within the base-paired subsequence is incorporated through the Q^b element(s).



$$R^{c}_{\textsc{Multi}}(i,j,\phi) \equiv \bigoplus_{L,R\in \mathrm{VP}(i,j,\phi)} \textsc{dot}(\{Q^{b}_{i,L}, Q^{b}_{R,j}, W(\Delta G^{\mathrm{coax}}_{i,L,j}(\phi))\}) \oplus \bigoplus_{\substack{k\in\{0,1\}\\ l\in\{0,1\}}} \begin{cases} Q^{b}_{i+k,j-l} \otimes W(\Delta G^{\mathrm{dangle}}_{i,i+k,j-l,j}(\phi), & \mathrm{FS}(i+k, j-l, \phi) \\ \mathbb{0}, & \text{otherwise} \end{cases} \tag{A.48}$$

Figure A.23: Polymer graph recursion diagram and recursion equation for the interstrand R^{c} recursion.

Both dangle and coaxial stacking states appear within multiloop contexts, albeit with additional multiloop-specific energy contributions. The multiloop equivalent of R^{c} is split into two separate recursions: R^{c_m}, which handles the dangle states, and R^{c_s}, which handles the coaxial stacking states. The recursion for $R^{c_m}_{\textsc{Multi}}(i,j,\phi)$ is shown in Figure A.24 and Equation A.49. The free energy contribution from the dangle state is handled in the same way as the analogous states included in R^{c}. Because the dangling is occuring in a multiloop context, we also account for the multiloop branch penalty for base pair $i+k \cdot j-l$ with $\Delta G^{\mathrm{multi}}_{\mathrm{bp}}$. Additionally, for whichever nucleotides are dangling in each state, these unpaired nucleotides are penalized appropriately with an equal number of $\Delta G^{\mathrm{multi}}_{\mathrm{nt}}$ terms. The dangling nucleotides $i+k$ and $j-l$ are required to be on the same strands as i and j respectively are required here to prevent exterior loop contributions in this multiloop context.

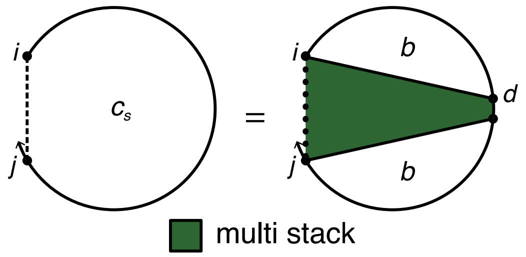


$$R^{c_m}_{\textsc{Multi}}(i,j,\phi) \equiv \bigoplus_{\substack{k\in\{0,1\}\\ l\in\{0,1\}}} \begin{cases} C_1, & \mathrm{FS}(i+k, j-l, \phi) \\ \mathbb{0}, & \text{otherwise} \end{cases} \tag{A.49}$$

$$\text{where } C_1 \equiv Q^b_{i+k,j-l} \otimes W(\Delta G^{\text{dangle}}_{i,i+k,j-l,j}(\phi) + \Delta G^{\text{multi}}_{\text{bp}} + (k+l)\Delta G^{\text{multi}}_{\text{nt}})$$

Figure A.24: Polymer graph recursion diagram and recursion equation for the interstrand R^{c_m} recursion.

The recursion $R^{c_s}_{\textsc{Multi}}(i,j,\phi)$, shown in Figure A.25 and Equation A.50, handles the coaxial stacks in multiloops in an analogous way to R^c handling these states in exterior loops. The same function $\Delta G^{\text{coax}}_{i,d,j}(\phi)$ is used to look up the appropriate coaxial stack free energy to include. The only difference is that R^{c_s} accounts for the multiloop branch penalties for base pairs $i \cdot d$ and $d+1 \cdot j$ with two counts of $\Delta G^{\text{multi}}_{\text{bp}}$. As the base pairs $i \cdot d$ and $d+1 \cdot j$ perfectly bisect the sequence $[i, j]$ into Q^b elements, no $\Delta G^{\text{multi}}_{\text{nt}}$ terms are needed to account for unpaired bases. Again, d and $d+1$ are required to be on the same strand to avoid exterior loops in this multiloop-contribution recursion.

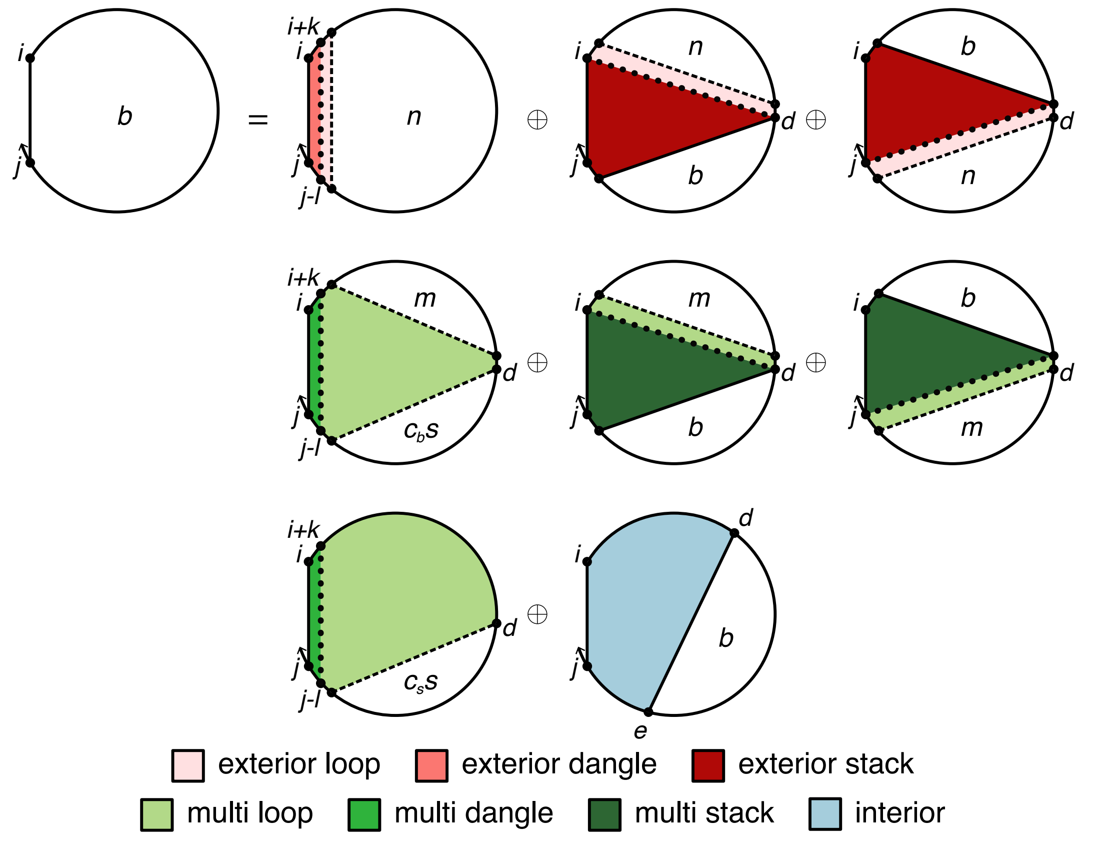


$$R^{c_s}_{\textsc{Multi}}(i,j,\phi) \equiv \bigoplus_{L,R\in \mathrm{VP}(i,j,\phi)} \textsc{dot}(\{Q^b_{i,L}, Q^b_{R,j}, W(\Delta G^{\text{coax}}_{i,L,j}(\phi))\}) \otimes W(2\Delta G^{\text{multi}}_{\text{bp}}) \tag{A.50}$$

Figure A.25: Polymer graph recursion diagram and recursion equation for the interstrand R^{c_s} recursion.

The recursion $R^b_{\textsc{Multi}}(i,j,\phi)$, shown in Figure A.26 and Equation A.51, is the most complex recusion: it handles interior loop contributions as well as multiloop and exterior loop coaxial and dangle stacking contributions. For interior loops, this recursion uses the same functions

Multi $O(N^4)$ Interior(i, j, ϕ) and Multi $O(N^3)$ Interior(i, j, ϕ) discussed in Section A.3.3. Because of the similarity of some of the recursion contributions for multiloop stacking states to exterior loop stacking states, they have been grouped in the subroutines Multi Closing Pair Stacking and Multi Closing Pair Dangles.



$$
\begin{aligned}
R^b_{\text{Multi}}(i, j, \phi) \equiv\ & \text{Multi Closing Pair Stacking}(i, j, \phi) \\
& \oplus \text{Multi Closing Pair Dangles}(i, j, \phi) \qquad \text{(A.51)} \\
& \oplus \text{Multi Interior}(i, j, \phi)
\end{aligned}
$$

Figure A.26: Polymer graph recursion diagram and recursion equation for the interstrand R^b recursion.

The subroutine Multi Closing Pair Stacking, shown in Equation A.52, handles the states where a base pair adjacent to $i \cdot j$ in $[i+1, j-1]$ (either $i+1$ or $j-1$ base paired) stacks on $i \cdot j$. These cases are depicted in Figure A.51 by the rightmost two recursion diagrams in the each of the first two rows. Of these, the recursion diagrams in the top row show this closing pair stacking in an exterior loop context, while the second row shows the stacking in a multiloop context. These cases must be considered in $R^b_{\text{Multi}}(i, j, \phi)$ separately from coaxial stacks occurring wholly in the sequence $[i+1, j-1]$ as this is the first time $i \cdot j$ is encountered. As the recursion collects the free energy of the coaxial stack between $i \cdot j$ and the adjacent base pair, the contribution from the sequence within the adjacent base pair is incorporated through the dependent $Q^b_{i+1,d-1}$ or $Q^b_{d,j-1}$ element. For multiloop contexts, the remainder of the contribution is incorporated through the $Q^m_{d,j-1}$ or $Q^m_{i+1,d-1}$ elements, respectively. For exterior loop contexts, the remainder of the contribution is incorporated through the $Q^n_{d,j-1}$ or $Q^n_{i+1,d-1}$ elements, respectively. The explicit coaxial stack energy is calculated with the functions $\Delta G^{\text{coax}}_{j,i,d-1}(\phi)$ or $\Delta G^{\text{coax}}_{d,j-1,i}(\phi)$, respectively. For the multiloop context, we are explicitly considering one closing base pair and one other pair, we account for the multiloop formation penalty, $\Delta G^{\text{multi}}_{\text{init}}$, and a branch penalty, $\Delta G^{\text{multi}}_{\text{bp}}$, for each pair. There is no equivalent of this in the exterior loop context. For both contexts, d and $d+1$ must be on the same strand. If this were not so, the multiloop states would degenerate to exterior loop states, and the exterior loop states would degenerate to disconnected states. Finally, there are cases, where Q^n elements cannot be used because the nick is adjacent to the inner base pair (d is at the $5'$ end of a strand or $d-1$ is at the $3'$ end), adjacent to i, or adjacent to j. These are handled in the contributions C_1, C_2, and C_3 in Equation A.52, respectively, which all use $Q^{\varnothing}$ elements in place of Q^n, because the nick is being handled at this level of recursion. The contributions C_2 and C_3 also include limiting states where no $Q^{\varnothing}$ element is accessed because the second base pair is $i+1 \cdot j-1$, leaving no extra subsequence to account for. As these states are all limiting cases of the last two diagrams in the first row of Figure A.26, their explicit polymer graph recursion diagrams were excluded for brevity.

$$\textsc{Multi Closing Pair Stacking}(i,j,\phi) \equiv \bigoplus_{L,R\in \mathrm{VP}(i+1,j-1,\phi)} C_0(L,R)$$
$$\oplus \begin{cases} C_1, & \mathrm{FS}(i+1,j-1,\phi) \\ C_2, & i+1=m \text{ and } j-1 \geq n \\ C_3, & j=n \text{ and } i+1<m \\ \mathbb{0}, & \text{otherwise} \end{cases}$$

$$\begin{aligned}
\text{where } m &= \textsc{First}(\eta) \\
n &= \textsc{Last}(\eta) \\
C_0(L,R) &\equiv \textsc{dot}(\{Q^b_{i+1,L}, Q^m_{R,j-1}, W(\Delta G^{\mathrm{coax}}_{j,i,L}(\phi))\}) \otimes W(\Delta G^{\mathrm{multi}}_{\mathrm{init}} + 2\Delta G^{\mathrm{multi}}_{\mathrm{bp}}) \\
&\oplus \textsc{dot}(\{Q^m_{i+1,L}, Q^b_{R,j-1}, W(\Delta G^{\mathrm{coax}}_{R,j-1,i}(\phi))\}) \otimes W(\Delta G^{\mathrm{multi}}_{\mathrm{init}} + 2\Delta G^{\mathrm{multi}}_{\mathrm{bp}}) \\
&\oplus \textsc{dot}(\{W(\Delta G^{\mathrm{coax}}_{j,i,L}(\phi)), Q^b_{i+1,L}, Q^n_{R,j-1}\}) \\
&\oplus \textsc{dot}(\{W(\Delta G^{\mathrm{coax}}_{R,j-1,i}(\phi)), Q^n_{i+1,L}, Q^b_{R,j-1}\}) \\
C_1 &\equiv \bigoplus_{d\in\eta} \left(W(\Delta G^{\mathrm{coax}}_{j,i,d-1}(\phi)) \otimes Q^b_{i+1,d-1} \otimes Q^{\varnothing}_{d,j-1} \oplus W(\Delta G^{\mathrm{coax}}_{d,j-1,i}(\phi)) \otimes Q^{\varnothing}_{i+1,d-1} \otimes Q^b_{d,j-1}\right) \\
C_2 &\equiv W(\Delta G^{\mathrm{coax}}_{i+1,j-1,i}(\phi)) \otimes Q^b_{i+1,j-1} \\
&\oplus \bigoplus_{L,R\in \mathrm{VP}(i,j-1,\phi)} \textsc{dot}(\{W(\Delta G^{\mathrm{coax}}_{R,j-1,i}(\phi)), Q^{\varnothing}_{i+1,L}, Q^b_{R,j-1}\}) \\
C_3 &\equiv W(\Delta G^{\mathrm{coax}}_{j,i,j-1}(\phi)) \otimes Q^b_{i+1,j-1} \\
&\oplus \bigoplus_{L,R\in \mathrm{VP}(i+1,j,\phi)} \textsc{dot}(\{W(\Delta G^{\mathrm{coax}}_{j,i,L}(\phi)), Q^b_{i+1,L}, Q^{\varnothing}_{R,j-1}\})
\end{aligned} \tag{A.52}$$

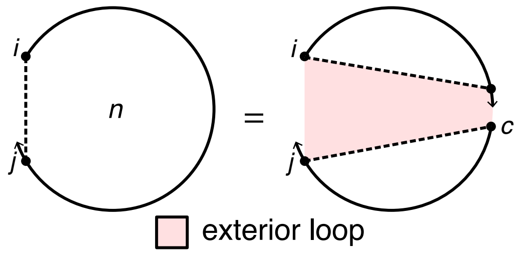
The subroutine Multi Closing Pair Dangles, shown in Equation A.53, handles the remaining multiloop states in which the base pair $i \cdot j$ does not coaxially stack with other other base pairs within the multiloop or exterior loop it bounds. Unpaired nucleotides in the loop may, however, dangle stack on base pair $i \cdot j$. For all these cases, Figure A.51 depicts these cases in the first column of contribution recursion diagrams. The top diagram reflects exterior loop states involving dangle stacking on $i \cdot j$. In all of these cases, the dangling nucleotides must be on the same strand as the adjacent base pair nucleotide to be valid. Generally, these cases incorporate the remaining sequence information $Q^n_{i+k+1,j-l-1}$, as depicted and shown as the last contribution of C_1 in Equation A.53. However, in cases where one of $i+k$ or $j-l$ is at the end of a strand, a Q^n cannot account for this nick and the form of the contribution is modified. These cases are handled by the contribution C_2, which uses $Q^{\varnothing}_{i+k+1,j-l-1}$ because the nick is explicitly known at this level of recursion. In the limiting case of this, there is only one nick such that $i+k+1=j-l$, and in this case the entire subsequence is accounted for an no $Q^{\varnothing}$ element is accessed, as in contribution C_3. These limiting cases are not depicted in Figure A.26

$$\textsc{Multi Closing Pair Dangles}(i, j, \phi) \equiv \bigoplus_{\substack{k\in\{0,1\}\\ l\in\{0,1\}}} \begin{cases} C_1, & \text{FS}(i+k+1, j-l-1, \phi) \\ C_2, & \text{FS}(i+k, j-l-1, \phi) \text{ and } i+k+1 = m \\ C_2, & \text{FS}(i+k+1, j-l, \phi) \text{ and } j-l = n \\ C_3, & i+k+1 = m \text{ and } j-l = n \text{ and } m = n \\ \mathbb{0}, & \text{otherwise} \end{cases}$$

$$\begin{aligned}
\text{where } b &= \eta \\
m &= \textsc{First}(b) \\
n &= \textsc{Last}(b) \\
\Delta G &\equiv \Delta G^{\text{dangle}}_{j-l,j,i,i+k}(\phi) \\
\Delta G^{\text{multi}}_{\text{closing}} &\equiv \Delta G^{\text{multi}}_{\text{init}} + \Delta G^{\text{multi}}_{\text{bp}} \\
C_1 &\equiv \textsc{dot}(\{Q^{c_s s}_{[i+k+1:m-1],j-l-1}, W(\Delta G^{\text{multi}}_{\text{closing}} + \Delta G^{\text{multi}}_{\text{nt}}[k+l : m-i+l-2])\}) \otimes W(\Delta G) \\
&\oplus \bigoplus_{L,R\in \text{VP}(i+k+1,j-l-1,\phi)} \textsc{dot}(\{Q^{m}_{i+k+1,L}, Q^{c_b s}_{R,j-l-1}\}) \otimes W(\Delta G^{\text{multi}}_{\text{closing}} + (k+l)\Delta G^{\text{multi}}_{\text{nt}} + \Delta G) \\
&\oplus W(\Delta G) \otimes Q^{n}_{i+k+1,j-l-1} \\
C_2 &\equiv Q^{\varnothing}_{i+k+1,j-l-1} \otimes W(\Delta G) \\
C_3 &\equiv W(\Delta G)
\end{aligned} \tag{A.53}$$

The multiloop contributions are handled in much the same way as in the single stranded recursion R^b. The middle diagram is roughly analogous to the way multiloop contributions are handled in Figure A.9. However, instead of splitting off all $3'$−most base pairs, this recursion splits off all $3'$−most multiloop stacking states with contribution $Q^{c_b s}_{d,j-l-1}$. The $R^{c_b s}$ recursion, discussed below, captures both single base pairs as well as coaxial stacking states (two base pairs) in a multiloop context. The remainder of the multiloop contribution is accounted for through the element $Q^{m}_{i+k+1,d-1}$. Only positions of d and $d-1$ on the same strand are valid to ensure the multiloop does not become an exterior loop. The bottom diagram depicts all states in an edge case where the are exactly three base pairs and the two subsequence base pairs coaxially stack on each other. This is reflected in needing to accumulate the explicitly coaxial stack information from $Q^{c_s s}$ instead of $Q^{c_b s}$, which has both coaxial and dangle stacking information. In both sets of cases, dangle stacking on base pair $i \cdot j$ is handled in the same way discussed for the recursion R^c. The only difference for these states is that the base pair itself is fixed while k and l determine whether the adjacent nucleotides stack or not. Any dangling nucleotides and all of the explicitly unpaired nucleotides in the latter type of contribution are accounted for by the correct number of multiloop unpaired nucleotide penalties, $\Delta G^{\text{multi}}_{\text{nt}}$. Additionally, the requirement that d be on the first strand prevents the multiloop state from degenerating into an exterior loop state handled elsewhere.

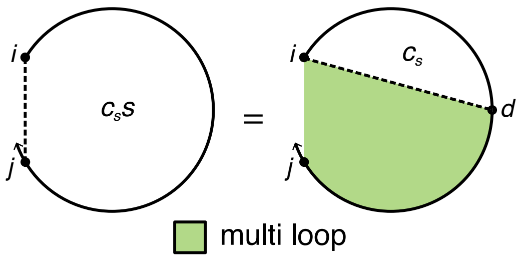
The recursion $R^{n}_{\textsc{Multi}}(i, j, \phi)$, shown in Figure A.27 and Equation A.54, caches the states where exactly one nick is explicitly exposed. The n can be thought of as standing for "nicked" contributions. It considers every nucleotide $c \in \eta$ that is $3'$ to a nick and incorporates the contributions for the two subsequences bisected by c through $Q^{\varnothing}_{i,c-1}$ and $Q^{\varnothing}_{c,j}$. On its own, all of the states it includes are disconnected because there is the nick c within the subsequence and no base pairs between the first and last strands in the block. However, when used in the $R^{b}_{\textsc{Multi}}(i, j, \phi)$ recursion where i and j are base paired and the other conditions of the contributions prevent other nicks from appearing, the state is connected with exactly one nick in $[i, j]$.



$$R^{n}_{\textsc{Multi}}(i, j, \phi) \equiv \bigoplus_{c \in \eta} Q^{\varnothing}_{i,c-1} \otimes Q^{\varnothing}_{c,j} \tag{A.54}$$

Figure A.27: Polymer graph recursion diagram and recursion equation for the R^{n} recursion.

The multiloop contributions for the recursion R^{b} are dependent on three recursions, $R^{c_s s}$, $R^{c_b s}$, and R^{m}, which have not yet been discussed. We describe them in the above order, which is from least to most dependent. The recursion $R^{c_s s}_{\textsc{Multi}}(i, j, \phi)$, shown in Figure A.28 and Equation A.55, acts as an efficiency wrapper for elements of Q^{c_s}. It collects contributions from all coaxial stacking states starting at i and ending in $[n, j]$. The lower bound on d prevents unaccounted for nicks that would lead to an exterior loop. The contributions from the individual coaxial stacks are incorporated from $Q^{c_s}_{i,d}$. This is combined with the contribution from the unpaired bases, $\Delta G^{\text{multi}}_{\text{nt}}(j - d)$. No branch penalties are applied because they have been handled already in R^{c_s}.

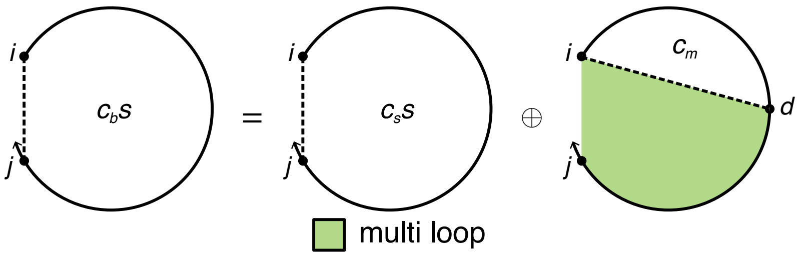


$$R^{c_s s}_{\textsc{Multi}}(i, j, \phi) \equiv \textsc{dot}(\{Q^{c_s}_{i,[n:j]}, W(\Delta G^{\text{multi}}_{\text{nt}}[0 : j - n]^r\}) \tag{A.55}$$

$$\text{where } n = \textsc{Last}(\eta)$$

Figure A.28: Polymer graph recursion diagram and recursion equation for the interstrand $R^{c_s s}$ recursion.

The recursion $R^{c_b s}_{\textsc{Multi}}(i, j, \phi)$, shown in Figure A.29 and Equation A.56, acts as an efficiency wrapper for elements of both Q^{c_s} and Q^{c_m}. That is, it captures multiloop stacking states starting at i and ending in $[n, j]$. Since the contributions from the multiloop coaxial stacking states have already been accumulated in $Q^{c_s s}_{i,j}$, they are captured here through that element. Then the dangle states starting at i and ending in $[n, j]$ are handled in an analogous way. The contributions from the individual dangle states are incorporated from $Q^{c_m}_{i,d}$. This is combined with the contribution from the unpaired bases, $\Delta G^{\text{multi}}_{\text{nt}}(j - d)$. No branch penalties are applied because they have already been handled in R^{c_m}. In both cases, the lower bound on d prevents nicks in $[d : j]$ that would lead to an exterior loop.



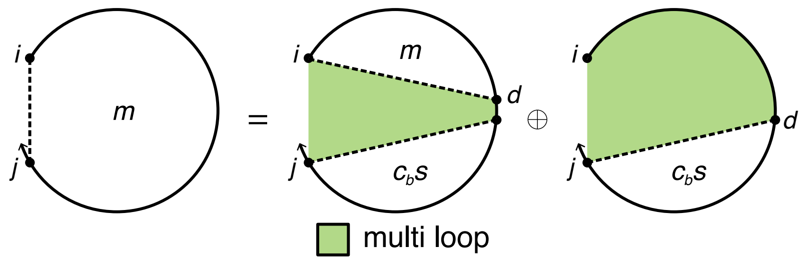
$$R^{c_b s}_{\textsc{Multi}}(i, j, \phi) \equiv Q^{c_s s}_{i,j} \oplus \textsc{dot}(\{Q^{c_m}_{i,[n:j]}, W(\Delta G^{\text{multi}}_{\text{nt}}[0 : j - n]^r\}) \tag{A.56}$$

$$\text{where } n = \textsc{Last}(\eta)$$

Figure A.29: Polymer graph recursion diagram and recursion equation for the interstrand $R^{c_b s}$ recursion.

The recursion $R^{m}_{\textsc{Multi}}(i, j, \phi)$, shown in Figure A.30 and Equation A.57, captures the recursive "remainder" of the multiloop contribution for use in R^b. There are two cases for how this remainder can look: one additional multiloop stacking state or more than one of these multiloop stacking states.

For multiple stacking states, the $3'$−most stacking state is incorporated explicitly through the $Q^{c_b s}_{d+1,j}$ while the remainder is handled through a recursive reference to a smaller subsequence value of $Q^{m}_{i,d}$. No explicit multiloop, dangle, or coaxial stack free energies need be considered because they have been handled in the dependent elements. Otherwise, there is exactly one additional stacking state in the multiloop, which is incorporated here through $Q^{c_b s}_{d,j}$ along with multiloop unpaired nucelotide penalties for the explicitly unpaired nucleotides in $[i, d-1]$. Here, the upper bound of m for d ensures d is always on the first strand, as otherwise there would be an exterior loop.



$$R^{m}_{\textsc{Multi}}(i,j,\phi) \equiv \bigoplus_{L,R\in \mathrm{VP}(i,j,\phi)} \textsc{dot}(\{Q^{m}_{i,L}, Q^{c_b s}_{R,j}\})$$
$$\oplus\ \textsc{dot}(\{Q^{c_b s}_{[i:m],j}, W(\Delta G^{\text{multi}}_{\text{nt}}[0 : m - i - 1])\}) \qquad (A.57)$$

where $m = \textsc{First}(\eta)$

Figure A.30: Polymer graph recursion diagram and recursion equation for the interstrand R^m recursion.

A.5 Evaluation algebras for each physical quantity

As discussed in the main text, we abstract the uses of recursions for different quantities using the concept of an *evaluation algebra*. An evaluation algebra $\mathcal{A}$ possesses the following concepts:

1. $D_{\mathcal{A}}$ is the domain of values in the evaluation algebra.

2. $\oplus_{\mathcal{A}}$ is an operation yielding a combination of alternative conditional ensembles. Thus, $c = a \oplus_{\mathcal{A}} b$ reflects the notion of "c contains either conditional ensemble a or b".

3. $\otimes_{\mathcal{A}}$ is an operation which yields a composition of conditional ensembles. Thus, $c = a \otimes_{\mathcal{A}} b$ reflects the notion of "c contains both conditional ensembles a and b".

4. $\mathbb{0}_{\mathcal{A}}$ is a value in $D_{\mathcal{A}}$ which satisfies the concept of additive identity. Physically, $\mathbb{0}_{\mathcal{A}}$ represents an impossible structure.

5. $\mathbb{1}_{\mathcal{A}}$ is a value in $D_{\mathcal{A}}$ which satisfies the concept of multiplicative identity. Physically, $\mathbb{1}_{\mathcal{A}}$ represents a structure in the free energy reference state.

6. $W_{\mathcal{A}}$ is an operation which yields a value in $D_{\mathcal{A}}$ from a secondary structure model free energy. Physically, $W_{\mathcal{A}}$ represents the Boltzmann weight on an individual substructure.

7. $Q_{\mathcal{A}}$ is an operation which yields a value in $D_{\mathcal{A}}$ from an recursion element in the set of all recursion elements Λ. $Q_{\mathcal{A}}$ is used to give the prior-calculated result over a given conditional ensemble.

As such, an evaluation algebra may be classified as an algebraic semiring equipped with two additional unary operators W and Q. As in the main text, we elide the dependence on $\mathcal{A}$ below for clarity when $\mathcal{A}$ is either given or interchangeable.

A.5.1 Evaluation algebras for scalar outputs

For completeness, we expand upon Table 2.2 by defining each evaluation algebra concept below. We catalogue the operation definitions used for each physical quantity which satisfy the requirements of an evaluation algbera. For the cases of secondary structure calculation (from Boltzmann sampling or the suboptimal structure ensemble), we elaborate our discussion below in Section A.5.2.

SumProduct: **sum product evaluation algebra.** Within evaluation algebra $\mathcal{A}$ = SumProduct, the partition function of an ensemble is computed by taking the sum over products of Boltzmann factors. Each expression in the algebra represents a Boltzmann factor, which is necessarily a non-negative real number. An impossible structure maps to a Boltzmann factor of 0, whereas a zero free energy structure (commensurate with the reference state adopted in the ensemble) maps to a Boltzmann factor of 1.

$$\begin{aligned}
D &= \mathbb{R}_{\geq 0} \\
a \oplus b &= a + b \\
a \otimes b &= a \cdot b \\
\mathbb{0} &= 0 \\
\mathbb{1} &= 1 \\
W(g) &= \exp\left(\frac{-g}{k_B T}\right) \\
Q(\lambda) &= A_{i,j} \text{ where } \lambda = (A, i, j) \text{ and } A \text{ is the stored recursion matrix}
\end{aligned} \tag{A.58}$$

Count: **structure count evaluation algebra.** Within evaluation algebra $\mathcal{A}$ = StructureCount, the number of structures within an ensemble is computed by taking the sum over products of subensemble counts. It may be easily seen that StructureCount is equivalent to SumProduct with the replacement of W. The domain of StructureCount is theoretically non-negative integers, but it is still implemented using floating point types to avoid integer overflow.

$$\begin{aligned} D &= \mathbb{Z}_{\geq 0} \\ a \oplus b &= a + b \\ a \otimes b &= a \cdot b \\ \mathbb{0} &= 0 \\ \mathbb{1} &= 1 \\ W(g) &= 1 \\ Q(\lambda) &= A_{i,j} \text{ where } \lambda = (A, i, j) \text{ and } A \text{ is the stored recursion matrix} \end{aligned} \tag{A.59}$$

MinSum: **minimum sum evaluation algebra.** Within evaluation algebra $\mathcal{A}$ = MinSum, the minimum free energy of an ensemble is the minimum over sums of conditional ensemble free energies. This evaluation algebra builds on the operations of the tropical semiring found in other contexts. The possibility of an impossible structure is included by assigning it a free energy of ∞.

$$\begin{aligned} D &= \mathbb{R} \cup \{\infty\} \\ a \oplus b &= \min(a, b) \\ a \otimes b &= a + b \\ \mathbb{0} &= \infty \\ \mathbb{1} &= 0 \\ W(g) &= v \\ Q(\lambda) &= A_{i,j} \text{ where } \lambda = (A, i, j) \text{ and } A \text{ is the stored recursion matrix} \end{aligned} \tag{A.60}$$

SplitExponent: **overflow-safe evaluation algebra.** We now expand our description in Table 2.2 of the overflow-safe evaluation algebra $\mathcal{A}$ = SplitExponent that we implemented. For exposition, that description included a free parameter γ representing the negative exponent of the output variable. Each Boltzmann factor is then evaluated relative to γ. To factor out γ, we lift our evaluation algebra into a set of higher order functions. Thus, instead of each expression being a pair of numbers, each expression is itself a function returning its associated mantissa value and its offset exponent relative to the input γ. We use the anonymous form of function notation $x \mapsto y$ to notate a function taking x and returning y.

An element a returns, as a function of γ, a pair of mantissa and exponent values, expressed respectively as $a_\mathrm{m}(\gamma)$ and $a_\mathrm{e}(\gamma)$ below. An element a may be converted to the domain of SumProduct using the transformation $a_\mathrm{m}(\gamma)2^{a_\mathrm{e}(\gamma)-\gamma}$. Theoretically, we can plug in any value of $\gamma = \gamma_0$ to calculate a desired result. Numerically, however, γ must be judiciously chosen to avoid floating point overflow. This choice of γ_0 is described in Section A.6.3.

$$
\begin{aligned}
D &= \mathbb{Z} \mapsto \mathbb{R}_{\geq 0} \times \mathbb{Z} \\
a \oplus b &= \gamma \mapsto \left(a_{\mathrm{m}}(\gamma) \cdot 2^{a_{\mathrm{e}}(\gamma)} + b_{\mathrm{m}}(\gamma) \cdot 2^{b_{\mathrm{e}}(\gamma)}, 0\right) \\
a \otimes b &= \gamma \mapsto (a_{\mathrm{m}}(b_{\mathrm{e}}(\gamma)) \cdot b_{\mathrm{m}}(\gamma), a_{\mathrm{e}}(b_{\mathrm{e}}(\gamma))) \\
\mathbb{0} &= \gamma \mapsto (0, 0) \\
\mathbb{1} &= \gamma \mapsto (1, \gamma) \\
W(g) &= \gamma \mapsto \left(\exp\left(\frac{-g}{k_B T}\right), \gamma\right) \\
Q(\lambda) &= \gamma \mapsto \left(M_{i,j}, E_{i,j} + \gamma\right) \text{ where } \lambda = (A, i, j) \text{ and } M, E \text{ are the} \\
&\qquad \text{stored recursion matrices for } A \text{ of mantissas and exponents, respectively}
\end{aligned}
\tag{A.61}
$$

In the above, addition works by aligning both expressions to the output exponent γ and then adding the resultant mantissas. As the output mantissa has been aligned to γ, the output shift exponent is 0. For example, take $a = \mathbb{1}$, $b = W(g_0)$. Then

$$
\begin{aligned}
a + b &= (a_{\mathrm{m}}(\gamma) \cdot 2^{a_{\mathrm{e}}(\gamma)} + b_{\mathrm{m}}(\gamma) \cdot 2^{b_{\mathrm{e}}(\gamma)}, 0) \\
&= \left(1 \cdot 2^{\gamma} + \exp\left(\frac{-g_0}{k_B T}\right) \cdot 2^{\gamma}, 0\right) \\
&= \left(2^{\gamma}\left(1 + \exp\left(\frac{-g_0}{k_B T}\right)\right), 0\right)
\end{aligned}
\tag{A.62}
$$

Meanwhile, multiplication works by multiplying the mantissas and adding the exponents. The exponent shift must be applied only to one quantity; therefore, the shift is applied directly to b, the result of which is propagated to a. For example, take $a = Q^b_{p,q}$, $b = Q^m_{r,s}$. Then

$$
\begin{aligned}
a \cdot b &= (a_{\mathrm{m}}(\gamma) \cdot b_{\mathrm{m}}(\gamma) \cdot M^m_{r,s}, a_{\mathrm{e}}(E^m_{r,s} + \gamma)) \\
&= (M^b_{p,q} \cdot M^m_{r,s}, E^b_{p,q} + E^m_{r,s} + \gamma)
\end{aligned}
\tag{A.63}
$$

LogSum: **log semiring evaluation algebra.** For completeness, we outline the possibility of using the log semiring LogSum to avoid overflow in partition function computation. In this evaluation algebra, each quantity x corresponds quantity 2^x in SumProduct (the base-2 logarithm is used for computational convenience.)

$$\begin{aligned}
D_{\textsc{LogSum}} &= \mathbb{R} \cup \{-\infty\} \\
\oplus_{\textsc{LogSum}}(a, b) &= \log_2(2^a + 2^b) = \max(a, b) + \log_2(2^{a-\max(a,b)} + 2^{b-\max(a,b)}) \\
\otimes_{\textsc{LogSum}}(a, b) &= a + b \\
\mathbb{0}_{\textsc{LogSum}}(a, b) &= -\infty \qquad\qquad (A.64)\\
\mathbb{1}_{\textsc{LogSum}} &= 0 \\
W_{\textsc{LogSum}}(g) &= -(kT \log 2)^{-1} g \\
Q(\lambda) &= A_{i,j} \text{ where } \lambda = (A, i, j) \text{ and } A \text{ is the stored recursion matrix}
\end{aligned}$$

In practice, this evaluation algebra proved to be simpler but less efficient than SplitExponent. Within a given dot product of many contributions, 1) the maximum contribution must be computed beforehand across all contributions, then 2) the adjusted exponentiations of each contribution must be calculated, then 3) the exponentiations must be summed. Even after optimization and vectorization, we found that LogSum was $> 2\times$ more expensive than SplitExponent in partition function computations due to the needs for floating point exponentiation and two separate scans through the arrays of contributions.

A.5.2 Evaluation algebras for structure generation

Structure generation conceptually yields specific secondary structures from a given weighting on the ensemble $\overline{\Gamma}$. In this case, because any given structure depends only on a sparse subset of recursion matrix elements, a backtracking operation order is in general more efficient than a forward pass iteration. Such an operation order jumps between recursion elements in an opposite direction to that in the forward pass. To enable such an approach, the recursion matrices in the forward pass must be computed beforehand: i.e. the minimum free energy before suboptimal structure generation and the partition function before sampling.

Here, we correspondingly distinguish between a *forward* evaluation algebra and a *backtracking* evaluation algebra. Whereas a forward evaluation algebra like SumProduct operates on a subset of $\mathbb{R}$, we define a backtracking evaluation algebra to operate on a domain of conditional ensembles which may be queried for a set of dependent recursion elements. This ordering may be viewed as equivalent to the topological ordering of the directed acyclic graph of computed quantities in a forward dynamic program (e.g. in Figure 2.8). In all cases, any conditional ensemble s containing an recursion element (b, i, j) indicates that $i \cdot j$ is a base pair in s, a feature which is used to output final structures from the algorithm. We next illustrate how a backtracking evaluation algebra may be defined with respect to the associated forward evaluation algebra.

Generic approach to structure generation. We start with a consideration of the simplest structure generation evaluation algebras. For conceptualization, in Table 2.2 we defined evaluation algebras ArgRandom to calculate a single randomly sampled structure and ArgMin to determine the MFE structure $s^{||}_{\mathrm{MFE}}$, assuming it is unique. For a given element a, each evaluation algebra was defined such that a was a pair of scalar value a_v and recursion element set a_λ.

As is clearly seen in Table 2.2, the operations on a_v and b_v in ArgRandom duplicate the operations of SumProduct, whereas the operations on a_v and b_v in ArgMin duplicate the operations of MinSum. Meanwhile, the operations on a_λ and b_λ within ArgRandom and ArgMin are almost identical. For instance, in both cases $\otimes$ represents the set union $\cup$ of recursion elements which occur in the same conditional ensemble.

The operations on a_λ and b_λ only diverge in the choice operator $\oplus$, which is responsible for attributing the scalar contribution $a_\mathrm{v} \oplus b_\mathrm{v}$ to an individual conditional ensemble a_λ or b_λ. In ArgRandom, $\oplus$ yields a random weighted choice via

$$\arg\mathrm{rand}(a, b) \equiv (a_\lambda \text{ if } \mathcal{U}(0, a_\mathrm{v} + b_\mathrm{v}) < a_\mathrm{v} \text{ else } b_\lambda) \tag{A.65}$$

where $\mathcal{U}$ is the random uniform distribution function. In contrast, in ArgMin, $\oplus$ yields the conditional ensemble which is lower in free energy via

$$\arg\min(a, b) \equiv (a_\lambda \text{ if } a_\mathrm{v} < b_\mathrm{v} \text{ else } b_\lambda)\,. \tag{A.66}$$

Thus we can see that an intuitive approach for constructing a backtracking evaluation algebra is to augment a corresponding forward evaluation algebra with customized operations for structure attribution. We catalogue the resultant definitions of ArgRandom and ArgMin below.

ArgRandom: **single Boltzmann sample evaluation algebra.** Within evaluation algebra $\mathscr{A}$ = ArgRandom, each element x is a pair of partition function value x_v and set of recursion elements x_λ. Operations on the first element x_v are defined using SumProduct. The second element x_λ is a set of recursion elements defining a restricted ensemble of conditional ensemble compositions (the set of all possible sets of recursion elements λ is denoted as $\mathcal{P}(\Lambda)$). Any quantity which does not depend on the output of another recursion is thus assigned $x_\lambda = \varnothing$.

$$\begin{aligned}
D &= \mathbb{R}_{\geq 0} \times \mathcal{P}(\Lambda) \\
a \oplus b &= (a_\mathrm{v} + b_\mathrm{v}, \arg\mathrm{rand}(a, b)) \\
a \otimes b &= (a_\mathrm{v} \cdot b_\mathrm{v}, a_\lambda \cup b_\lambda) \\
\mathbb{0} &= (0, \varnothing) \\
\mathbb{1} &= (1, \varnothing) \\
W(g) &= \left(\exp\left(\frac{-g}{k_B T}\right), \varnothing\right) \\
Q(\lambda) &= (Q_{\textsc{SumProduct}}(\lambda), \{\lambda\})
\end{aligned} \tag{A.67}$$

ArgMin: **unique MFE structure evaluation algebra.** Within evaluation algebra $\mathcal{A}$ = ArgMin, each element x is a pair of value x_v and set of recursion elements x_λ. Operations on the first element x_v are defined using MinSum. The second element x_λ is a set of recursion elements defining a restricted ensemble of conditional ensemble compositions (the set of all possible sets of recursion elements λ is denoted as $\mathcal{P}(\Lambda)$). Any quantity which does not depend on the output of another recursion is thus assigned $x_\lambda = \varnothing$.

$$\begin{aligned}
D &= \mathbb{R} \cup \{\infty\} \times \mathcal{P}(\Lambda) \\
a \oplus b &= (\min(a_\text{v}, b_\text{v}), \arg\min(a, b)) \\
a \otimes b &= (a_\text{v} + b_\text{v}, a_\lambda \cup b_\lambda) \\
\mathbb{0} &= (0, \varnothing) \\
\mathbb{1} &= (1, \varnothing) \\
W(g) &= \left(\exp\left(\frac{-g}{k_B T}\right), \varnothing\right) \\
Q(\lambda) &= (Q_{\text{MinSum}}(\lambda), \{\lambda\})
\end{aligned} \tag{A.68}$$

Efficient structure generation via lazy evaluation. We derived more programmatically efficient evaluation algebras for Boltzmann sampling, MFE structure, and suboptimal structure generation. In the following subsections, we describe efficient evaluation algebras for ArgRandomN and ArgGap which build upon ArgRandom and ArgMin.

The simpler but less efficient algebras ArgRandom and ArgMin yield full representations of the chosen conditional ensembles, which are then enqueued by the respective operation order algorithms. Our more efficient algorithms work by backtracking through a given recursion element and enqueueing any combinations of recursion elements in conditional ensembles that match a given criterion. The matching evaluation algebras incorporate the enqueueing operation κ directly such that each piece of a conditional ensemble is enqueued immediately as it is encountered. These evaluation algebras are similarly generic but operate lazily on recursion elements (obviating storage of intermediate structures which might not affect the final results).

We describe the ArgRandomN and ArgGap evaluations using a generic framework defined with respect to a given forward algebra (SumProduct and MinSum respectively). As in Section A.5.1, we make use of the anonymous form of function notation $x \mapsto y$ to notate a function taking x and returning y. Formally, we define the enqueueing operation κ recursively as a function in $D_\kappa \equiv (\mathbb{R}, \mathcal{P}(\Lambda)) \mapsto D_\kappa$; effectively, it may be viewed as an iterator across each alternative conditional ensemble. Let $\mathcal{B}$ be the backward algebra and $\mathcal{A}$ the forward algebra. Then we classify each expression in the evaluation algebra as a closure within $D_\mathcal{A} \equiv D_\kappa \mapsto D_\kappa$ and denote a set of recursion elements as $\Lambda_i \in \mathcal{P}(\Lambda)$ below.

Within the evaluation algebra, addition of a and b intuitively represents the successive iteration through multiple alternative structures a and b. It may be defined formally as the higher-order function:

$$\oplus_{\mathcal{B}}(a, b) = \kappa \mapsto b(a(\kappa)) \tag{A.69}$$

Multiplication of a and b represents the independent combinations of conditional ensembles from a and being composed together–in effect, a lazily evaluated outer product of conditional ensembles within a and b. It may be defined formally as the higher-order function:

$$\otimes_{\mathcal{B}}(a, b) = \kappa \mapsto a\big((x_1, \Lambda_1) \mapsto b\big((x_2, \Lambda_2) \mapsto \kappa\big(x_1 \otimes_{\mathcal{A}} x_2, \Lambda_1 \cup \Lambda_2\big)\big)\big) \tag{A.70}$$

The properties of commutativity and associativity are preserved for $\oplus_{\mathcal{B}}$ and $\otimes_{\mathcal{B}}$ so long as κ is independent of the order of evaluation (i.e. $\kappa(x_1, \Lambda_1)(x_2, \Lambda_2) = \kappa(x_2, \Lambda_2)(x_1, \Lambda_1)$), a property which is satisfied by all algorithms discussed here. The multiplicative identity corresponds to a zero free energy structure, which is not dependent on any recursion elements:

$$\mathbb{1}_{\mathcal{B}} = \kappa \mapsto \kappa(\mathbb{1}_{\mathcal{A}}, \varnothing) \tag{A.71}$$

The additive identity is defined as the identity function, reflecting an impossible structure by doing nothing:

$$\mathbb{0}_{\mathcal{B}} = \kappa \mapsto \kappa \tag{A.72}$$

$W_{\mathcal{B}}$ brings a free energy parameter into the evaluation algebra domain. It is not dependent on any recursion elements either:

$$W_{\mathcal{B}}(g) = \kappa \mapsto \kappa\left(W_{\mathcal{A}}(g), \varnothing\right) \tag{A.73}$$

Finally, the recursion matrix operator yields the forward algebra value and a singleton of its associated recursion element:

$$Q_{\mathcal{B}}(\lambda) = \kappa \mapsto \kappa\left(Q_{\mathcal{A}}(\lambda), \{\lambda\}\right) \tag{A.74}$$

See Sections A.6.5 and A.6.6 for specifications of the enqueing function κ used in Boltzmann sampling and suboptimal structure generation, respectively.

ArgRandomN: **simultaneous Boltzmann sample evaluation algebra.** We here define the implemented evaluation algebra $\mathcal{A} = \textsc{ArgRandomN}$ which uses higher order functions to accomplish lazy evaluation as explained more fully in Section A.5.2. The below is simply a specialization of the generic structure generation algebra for the associated forward evaluation algebra SumProduct.

$$\begin{aligned} D &= D_\kappa \mapsto D_\kappa \\ a \oplus b &= \kappa \mapsto b(a(\kappa)) \\ a \otimes b &= \kappa \mapsto a\big((x_1, \Lambda_1) \mapsto b\big((x_2, \Lambda_2) \mapsto \kappa\big(x_1 \cdot x_2, \Lambda_1 \cup \Lambda_2\big)\big)\big) \\ \mathbb{0} &= \kappa \mapsto \kappa \\ \mathbb{1} &= \kappa \mapsto \kappa(1, \varnothing) \\ W(g) &= \kappa \mapsto \kappa\left(\exp\left(\frac{-g}{k_B T}\right), \varnothing\right) \\ Q(\lambda) &= \kappa \mapsto \kappa(Q_{\textsc{SumProduct}}(\lambda), \{\lambda\}) \end{aligned} \tag{A.75}$$

To avoid overflow issues on large sequences, we also extended the evaluation algebra above in a similar fashion to that described in Section A.5.1.

ArgGap: **suboptimal structure evaluation algebra.** We here define the implemented evaluation algebra $\mathcal{A} = \textsc{ArgGap}$ which uses higher order functions to accomplish lazy evaluation as explained more fully in Section A.5.2. The below is simply a specialization of the generic structure generation algebra for the associated forward evaluation algebra MinSum.

$$\begin{aligned} D &= D_\kappa \mapsto D_\kappa \\ a \oplus b &= \kappa \mapsto b(a(\kappa)) \\ a \otimes b &= \kappa \mapsto a\big((x_1, \Lambda_1) \mapsto b\big((x_2, \Lambda_2) \mapsto \kappa\big(x_1 + x_2, \Lambda_1 \cup \Lambda_2\big)\big)\big) \\ \mathbb{0} &= \kappa \mapsto \kappa \\ \mathbb{1} &= \kappa \mapsto \kappa(0, \varnothing) \\ W(g) &= \kappa \mapsto \kappa(g, \varnothing) \\ Q(\lambda) &= \kappa \mapsto \kappa(Q_{\textsc{MinSum}}(\lambda), \{\lambda\}) \end{aligned} \tag{A.76}$$

A.6 Operation orders for each physical quantity

A.6.1 A partial order on recursion elements

We developed novel dynamic programming algorithms to take advantage of the blockwise approach to calculations in the multistranded ensemble. The resultant dependency graph of recursions provides the main constraints in correctly handling calculations of a given physical quantity. To formalize this dependency structure, we define a partial order on recursions λ_p, λ_q such that if (and only if) $\lambda_p < \lambda_q$, then the definition of recursion λ_q is dependent on that for λ_p. We define this partial order as a lexicographical order on (1) the strand indices of the recursions, (2) the subsequence indices of the recursions, and (3) the recursion types. In other words, two recursions are to be compared based on their associated strand indices, then their subsequence indices, then their recursion types.

Ordering on strand indices. We developed dynamic programming algorithms working over subsequences of strands within a multistranded complex. Thus, we implement a partial order on the strand indices a, b of two recursions by defining that $\lambda_p < \lambda_q$ if ($a_q < a_p$ and $b_p \leq b_q$) or ($a_q < a_p$ and $b_p \leq b_q$). (We adopt the convention that the indices are sorted such that $a \leq b$ for any recursion.) Figure A.31 illustrates the dependency structure induced by recursion strand indices and its effect on the operation order used in our dynamic programs.

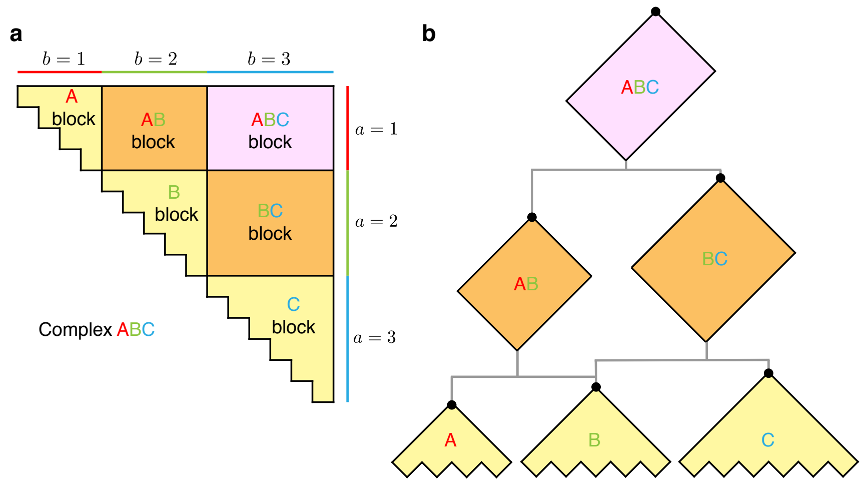


Figure A.31: Blockwise operation order. (a) Depiction of the blockwise approach. (b) Illustration of the induced dependency structure. Analogous to a single stranded dynamic program, which uses subproblems on subsequences of nucleotides, the multistranded dynamic program uses subproblems on subsequences of strands. Panel (a) gives examples of strand indices a and b used in the pseudocode of Sections A.6.2 and A.6.4. Panel (b) shows an example dependency graph between blocks, with the dependency direction depending on whether a forward or backtracking operation is being used. Black circles denote locations in the forward algorithms where block results may be cached to avoid recomputation.

Ordering on subsequence indices. Next we incorporate a partial order on subsequence indices by additionally defining that, if the strand indices of λ_p, λ_q are identical, then $\lambda_p < \lambda_q$ if ($i_q < i_p$ and $j_p \leq j_q$) or ($i_q < i_p$ and $j_p \leq j_q$). (We adopt the convention that the indices are sorted such that $i \leq j$ for any recursion.) This mirrors the structure encountered with strand indices and is the historical order implicit in dynamic programming algorithms working within a single-stranded ensemble.

Ordering on recursion types. Finally, we define a partial order on recursion types for when the strand and subsequence indices of λ_p and λ_q are identical. Let $\mathbf{T}$ be an ordered sequence of recursion types in a given set of recursions such that if for any integers i, j, p, q, if $p < q$ and $\lambda_p \equiv (\mathbf{T}_p, i, j)$ and $\lambda_q \equiv (\mathbf{T}_q, i, j)$ then $\lambda_p < \lambda_q$. In other words, for a given i, j, the recursion $(\mathbf{T}_p, i, j)$ is not dependent on that for $(\mathbf{T}_q, i, j)$. There are multiple logically consistent recursion type sequences $\mathbf{T}$, but we implemented the following ones:

$$\begin{aligned} \mathbf{T}_{\text{no stacking}} &\equiv [\text{X, MB, B, T, D, YA, YB, MS, M, S}, \varnothing] \\ \mathbf{T}_{\text{stacking}} &\equiv [\text{X, B, T, D, YA, YB, CM, CS, CSS, CBS, C, S, M}, \varnothing, \text{N}] \end{aligned} \tag{A.77}$$

A.6.2 Partition function, structure count, and MFE

We give the following algorithm for the block-based dynamic program over subcomplexes used for partition function, structure count, and MFE. It relies on separate subroutines to calculate triangular intrastrand blocks and rectangular interstrand blocks.

ComplexDynamicProgram takes parameters ϕ, the sequence of the complex for which to compute the partition function, and C, a map from sequences to blocks in which to store and retrieve computed blocks. It returns the complete block of dynamic program results FullQ.

```
ComplexDynamicProgram($\mathcal{A}$, $\phi$, $C$)
    $L$ = Number of Sequences($\phi$)
    FullQ $\leftarrow$ EmptyBlock(Length($\phi$)) // Initialize all matrix storage

    for $l \in [0 : L]$
        for $a \in [1 : L - l]$
            $b \leftarrow a + l$
            if $\phi^{a,b} \in C$
                // Take result for block from cache
                FullQ$_{a,b}$ $\leftarrow C[\phi^{a,b}]$
            else
                if $a = b$
                    // Calculate intrastrand block
                    FullQ$_{a,a}$ $\leftarrow$ IntraStrandDynamicProgram($\mathcal{A}$, $\phi^{a,a}$)
                else
                    // Calculate interstrand block
                    InterStrandDynamicProgram($\mathcal{A}$, $\phi^{a,b}$, FullQ$_{[a:b],[a:b]}$)
                // Put result for block into cache
                $C[\phi^{a,b}] \leftarrow$ FullQ$_{a,b}$
    return FullQ
```

Algorithm A.4: Blockwise operation order over subcomplexes

The outermost element of FullQ corresponding to $Q_{1,N}$ may be post-processed into the target physical quantity as described in Section A.7 and Reference 2.

Operation order for intrastrand blocks. We define the subsidiary operation order as follows for a single strand sub-complex to return a fresh block Q. No prior information from other blocks is necessary. Iteration proceeds as in Figure 2.5.

```
IntraStrandDynamicProgram($\mathcal{A}, \phi$)
    $N \leftarrow$ Length($\phi$)
    for $l \in [1 : N]$
        for $i \in [1 : N - l]$
            $j \leftarrow i + l - 1$
            for $T \in \mathbf{T}$
                // Calculate and store recursion output for $Q^T_{i,j}$
                $\lambda \leftarrow (T, i, j)$
                $Q(\lambda) \leftarrow R_{\text{Single}}(\lambda, \phi)$
    return $Q$
```

Algorithm A.5: Operation order for single stranded blocks

Operation order for interstrand blocks. We define the subsidiary operation order for a multi-strand sub-complex to update the parameter Q with the outermost block, given that all dependent blocks are already calculated. For instance, in Figure A.31(a), InterStrandDynamicProgram would calculate the top-right block ABC using the prior calculations of blocks A, B, C, AB, and BC.

```
InterStrandDynamicProgram($\mathcal{A}, \phi, Q$)
    $N \leftarrow$ Length($\phi$)
    $b \leftarrow$ Nicks($\phi$)
    $m \leftarrow$ First($b$) // index of first base on second strand
    $n \leftarrow$ Last($b$) // index of first base on last strand

    // Iteration proceeds from lowest to highest $l \equiv j - i + 1$
    for $l \in [n - m + 1 : N]$
        for $i \in [\max(l, n) - l + 1 : \min(m, N - l)]$
            $j \leftarrow i + l - 1$
            for $T \in \mathbf{T}$
                // Calculate and store recursion output for $Q^T_{i,j}$
                $\lambda \leftarrow (T, i, j)$
                $Q(\lambda) \leftarrow R_{\text{Multi}}(\lambda, \phi)$
```

Algorithm A.6: Operation order for multistranded blocks

A.6.3 Overflow-safe partition function

As described in Section A.5.1, overflow prevention during partition function or structure count computation can be acccomplished using a split representation of all quantities in terms of separate mantissas and exponents. Our overflow-safe algorithm uses the same operation order as A.6.2, just with a different evaluation algebra to calculate and store recursion element matrices.

Any expression within the SplitExponent evaluation algebra is a function of γ which returns the 1) mantissa x_{m} and 2) exponent x_{e} relative to that given exponent shift such that the value of the expression is $x_{\mathrm{m}} 2^{x_{\mathrm{e}}-\gamma}$ for any choice of γ. For each recursion matrix, mantissa and exponent values of the same bit width are held in separate arrays M and E. Single-precision floating point and signed integers are used, such that the total storage cost of this method is identical to running SumProduct in double precision. From the output expression of a given recursion R in SplitExponent, the respective entry in the recursion matrix is set via the following procedure:

$$\begin{aligned} M(\lambda) &\leftarrow R_{\mathrm{m}}(\lambda, \phi)(\gamma_0(\lambda)) \\ E(\lambda) &\leftarrow \gamma_0(\lambda, \phi) + R_{\mathrm{e}}(\lambda)(\gamma_0(\lambda)) \end{aligned} \tag{A.78}$$

As described in Section A.5.1, γ_0 must be chosen judiciously for a given computation. If an expression has a theoretical value of q, γ_0 should be relatively close to $-\log_2 q$ to prevent overflow from occurring. For recursion element $\lambda = (A, i, j)$, we choose γ_0 as follows from the matrix E containing the exponents of A:

$$\begin{aligned} \gamma_0(\lambda) &\equiv \min_{\substack{i \le d \le e \le j \\ (d,e) \neq (i,j)}} -E_{d,e} \\ &= \begin{cases} \min\left(-E_{i+1,j}, -E_{i,j-1}\right) & i < j \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{A.79}$$

Although it possible to try multiple choices of γ_0 as a failsafe, in practice the single definition of γ_0 above is sufficient to avoid overflow in the entries of M.

A.6.4 Pair probability matrix

We present pseudocode for a backtrack-free multistranded pair probabilities algorithm below. Symbols have the same meanings as in Section A.6.2, except FullQ is the block for the doubled sequence ϕ' instead of the input ϕ. Similarily, the Q and Q^b matrices in line 0 refer to the recursion matrices for ϕ'. After the dynamic programming algorithm the resultant Q and Q^b entries are post-processed into the pair probabilities matrix as in Equation 2.17.

The subroutine PartialInterStrandDynamicProgram behaves identically to InterStrandDynamicProgram but stops once l in its outer recursion reaches $N \equiv \text{Length}(\phi)$. This savings can take place because the backtrack-free pair probabilities methodology (Figure 2.13) only requires results from element indices (i, j) such that $i \leq j \leq i + N$.

```
PairProbabilities($\mathcal{A}$, $\phi$, FullQ, $C$)
    $N$ = Length($\phi$)
    $\phi' = \phi + \phi$
    Initialize FullQ to $\mathbb{0}$
    $L$ = Number of Sequences($\phi$)
    for $l \in [0 : L]$
        for $a \in [1 : 2L - l]$
            $b \leftarrow a + l$
            if $\phi'^{a,b} \in C$
                // Take result from cache
                FullQ$_{a,b} \leftarrow C[\phi'^{a,b}]$
            else
                if $a = b$
                    // Calculate intrastrand block
                    FullQ$_{a,a} \leftarrow$ IntraStrandDynamicProgram($\mathcal{A}$, $\phi'^{a}$)
                elseif $l < L$
                    // Calculate interstrand block
                    InterStrandDynamicProgram($\mathcal{A}$, $\phi'^{a,b}$, FullQ$_{[a:b],[a:b]}$)
                else
                    // Calculate lower triangle of interstrand block
                    PartialInterStrandDynamicProgram($\mathcal{A}$, $\phi'^{a,b}$, FullQ$_{[a:b],[a:b]}$, $N$)
                // Put results in cache
                $C[\phi'^{a,b}] \leftarrow$ FullQ$_{a,b}$

    Initialize $N \times N$ matrix $P$
    for $i \in [1 : N]$
        for $j \in [1 : N]$ and $j \neq i$
            $P_{i,j} = Q^b_{i,j} Q^b_{j,N+i} Q^{-1}_{1,N}$ // same as Equation 2.17
        $P_{i,i} = 1 - \sum_{1 \leq j \leq N, j \neq i} P_{i,j}$
    return $P$
```

Algorithm A.7: Operation order for backtrack-free pair probabilities.

A.6.5 Sampled structure generation

While the pair probability P and partition function Q of a given complex ensemble yield exact and important equilibrium statistics, other statistics must be estimated in a randomized fashion via sampled structures from the Boltzmann ensemble. Doing so enables determination of higher-order statistics including conditional base pairing information (e.g. in what fraction of structures do two regions of a complex appear as duplexes simultaneously?). Motivated by the central use-case of estimating equilibrium averages over Γ, we develop here novel methods generating J independently sampled structures from the same ensemble at a reduced runtime complexity. Previous algorithms have provided the ability to sample a single structure from the Boltzmann distribution for a complex, i.e. structure s sampled with probability $p(\phi, s)$. An algorithm from Ding and Lawrence[15] achieved a worst-case $O(JN^2)$ runtime for sampling single-stranded structures, beyond the $O(N^3)$ cost of computing the partition function. In this case, both complexity results were achieved by excluding interior loops with more than 30 unpaired nucleotides. NUPACK 3 included an algorithm for interstrand complexes based on a full backtrack through each element of the recursion matrices, incurring an $O(JN^3)$ runtime complexity without excluding any interior loops.

In contrast, by adapting the iteration order of the Q^b recursion such that interior loop contributions are considered in order from fewest to most unpaired bases, we first achieve a worst-case runtime of $O(JN^2)$ without exclusion of any interior loop states for sequential sampling. This complexity is roughly due to performing a scan of at most $O(N)$ contributions to each of the $O(N)$ recursion elements within each of the J structures. We improve upon this result by developing an exact, efficient algorithm for simultaneous sampling.

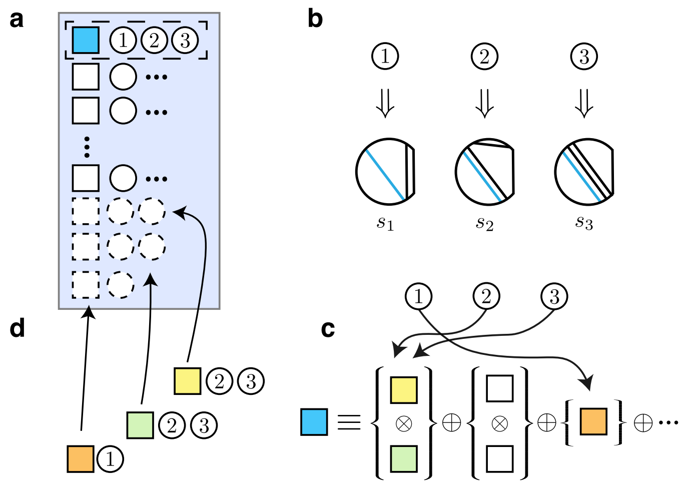


Figure A.32: Illustration of multisampling operation order. (a) The top recursion element is popped off the priority queue along with its associated structures 1,2,3. (b) If the popped element is of type $Q^b_{d,e}$, a base pair between d and e is added to each associated structure 1,2,3. (c) The evaluation algebra is invoked with $n = 3$, randomly assigning conditional ensembles to structures 1,2,3. (d) The recursion elements corresponding to each conditional ensemble is added to the priority queue along with its associated structures.

Algorithm overview. We developed a simultaneous sampling algorithm using a new operation order that eliminates any recomputations of the same recursion (Figure A.32). The priority queue is defined via the partial order on recursion element λ from Section A.6.1 via the recursion type, strand indices, and sequence indices of λ. The queue is initialized with the single recursion element $Q_{1,N}$ and the indices of the associated sampled structures $1, \ldots, J$ (i.e., all J sampled structures that are to be generated). When an element is popped from the priority queue, if J_λ of the sampled structures include this element, J_λ random numbers are drawn and sorted. Next, each of N_λ conditional ensembles is traversed exactly once, and each matching contribution is enqueued along with every index of a matched structure. This procedure yields a subproblem complexity of $O(N_\lambda + J_\lambda \log J_\lambda)$ compared to $O(J_\lambda N_\lambda)$.

The resultant performance of the entire sampling algorithm is worst-case bounded $O(JN^2)$ but otherwise sequence-dependent. The speedup from simultaneous sampling is expected to be greatest when the Boltzmann ensemble is dominated by only a few conditional ensembles. Still, the method achieves order-of-magnitude speedups across benchmark ensembles of random sequences (Figure 2.14, Section A.8.5) and designed sequences (Section A.8.5). Empirically, we observe near-best case performance across these ensembles with an empirical linear complexity in N and a sublinear complexity in J (around $O(J^{0.8})$). More performance details are given in Section A.8.5.

Operation order. Below we give an algorithm for simultaneously generating J secondary structures randomly from the Boltzmann distribution for a complex of sequence ϕ with $N \equiv |\phi|$.

1. Initialize an array L of J secondary structures with no base pairs.
2. Initialize an empty priority queue $\mathcal{P}$ of pairs of recursion recursion element λ and vector of ordered structure indices $\vec{v}$.
3. Enqueue a pair of $\lambda = (\varnothing, 1, N)$ and $\vec{v} = [1 : J]$ into $\mathcal{P}$.
4. While $\mathcal{P}$ is not empty:
   a) Dequeue the highest priority element λ and its respective indices $\vec{v}$ from $\mathcal{P}$ (Figure A.32(a)).
   b) If λ denotes any element $Q^b_{i,j}$, add a base pair $i \cdot j$ in each structure s_l for $l \in \vec{v}$ (Figure A.32(b)).
   c) Let J_λ be the length of $\vec{v}$. Initialize an array $\vec{w}$ of J_λ random numbers uniformly distributed between 0 and $Q_{\text{SumProduct}}(\lambda)$.
   d) Sort $\vec{w}$, incurring $O(J_\lambda \log J_\lambda)$ cost and reorder $\vec{v}$ by the same permutation.
   e) Initialize $q = 0$ as the running sum of contributions and $k = 1$ as the running index.
   f) Calculate the generator $G = R_{\text{ArgRandomN}}(\lambda)(\kappa)$ in order to attribute conditional ensemble contributions to the output structure indices $\vec{v}$ (Figure A.32(c)). For exposition, this may

be achieved with the coroutine $\kappa(x, \Lambda)$ which yields (x, Λ) and returns κ. In practice, the loop below was programmatically implemented via a callback function.

g) For each of N_λ contributions (x, Λ) in G until $k > J_\lambda$:

   i. Increment the accumulator $q \leftarrow q + x$.

   ii. Find the remaining weights below q by calculating $k' \leftarrow \textsc{UpperBound}(\vec{w}[k : j], q)$. This may be done via binary search; the yielded subproblem complexity from this step is still within $O(J_\lambda + N_\lambda)$.

   iii. For each element λ in Λ, enqueue $(\lambda, \vec{v}[k : k' - 1])$ into $\mathcal{P}$ (Figure A.32(d)). If λ was already present in $\mathcal{P}$, concatenate the indices $\vec{v}$ of the two items.

   iv. Update the running index $k \leftarrow k'$.

5. Return L.

We achieve an $O(N_\lambda + J_\lambda \log J_\lambda)$ in the subproblem of backtracking through a given recursion element λ (see the complexity annotations in Step 4 above). Empirical measurements of the complete algorithm complexity are given in Sections A.8.5 and A.8.5.

Recursion iteration order for interior loops in backtracking algorithms. We sample from the structural ensemble containing all interior loop states while achieving an asymptotic upper bound of $O(N^2)$ for a single sample. This is made possible by iterating through the interior loop states in order from fewest to most unpaired nucleotides, as discussed below. This iteration order is shown for intrastrand and interstrand recursions in Equations A.80 and A.81, respectively. This contrasts with the historical iteration order for interior loops, which considers all $5'$ inner bases d in ascending and then all $3'$ inner bases e compatible with each d, again in ascending order, shown in Equations A.28 and A.29.

Note that the iteration orders (A.80) and (A.81) result in $O(N^4)$ forward pass algorithms, as discussed in Section A.3.3 for (A.28) and (A.29). This occurs because for each closing pair $i \cdot j$, we consider all $O(n^2)$ possible second base pairs $d \cdot e$, where $n = j - i + 1$. However, as we will show below, this does not affect the $O(N^2)$ single sample performance we are claiming.

$$\textsc{Single Sampling Interior}(i, j, \phi) \equiv \bigoplus_{z=10}^{j-i-4} \bigoplus_{s=5}^{z-5} \{Q^b_{d,e} \otimes W(\Delta G^{\text{interior}}_{i,d,e,j}(\phi))\} \tag{A.80}$$

$$\text{where } d = i + z - s$$
$$e = j - s$$

$$\text{Multi Sampling Interior}(i, j, \phi) \equiv \bigoplus_{z=10}^{j-n+m-i} \bigoplus_{r=\max(5,z-j+n)}^{\min(z-5,m-i-1)} \{Q^b_{d,e} \otimes W(\Delta G^{\text{interior}}_{i,d,e,j}(\phi))\}$$

$$\begin{aligned} \text{where } d &= i + r \\ e &= j + r - z \\ m &= \text{First}(\eta) \\ n &= \text{Last}(\eta) \end{aligned} \tag{A.81}$$

We proceed to show that iterating through extensible interior loops in order from fewest to most unpaired bases results in an overall asymptotic upper bound of $O(N^2)$. In the recursions for sampling the contributions to an element $Q^b_{i,j}$, hairpin loops, exterior loops, multiloops, and inextensible interior loops (including all bulge loops and stack loops) are all sampled first. From the R^b recursions in Figures A.4, A.9, A.17, A.26, one can see there are either $O(1)$ or $O(n)$ of these contribution types for a subsequence of length n. There is only one hairpin loop, one stack loop, and $O(n)$ bulge and inextensible interior loops. While there are potentially more than $O(n)$ multiloops consistent with $i \cdot j$, they are handled recursively and there are only $O(n)$ contributions coming through Q^m elements. Therefore, if only these states are sampled, the algorithm will only recurse into at most $O(N)$ Q^b elements each costing at most $O(N)$ for an over all complexity of $O(N^2)$ and we would already have our bound.

So we can limit ourselves to cases where at least one extensible interior loop is sampled. If iteration proceeds through these interior loops in ascending order of number of unpaired bases, each inner base pair $d \cdot e$ will be encountered at most once. To see this, assume the extensible interior loop with base pair $d \cdot e$ is sampled. Then every previous base pair $d' \cdot e'$ iterated through in order to reach $d \cdot e$ will meet one of the following conditions: $e' - d' > e - d$ or $d' < d < e' < e$. The first case occurs for all interior loops with fewer unpaired nucleotides than the loop bounded by base pair $d \cdot e$. The second case occurs for all interior loops with the same number of unpaired nucleotides as the loop bounded by base pair $d \cdot e$. In both cases, $\phi_{d',e'}$ is not a subsequence of $\phi_{d,e}$ and the base pair $d' \cdot e'$ cannot appear in $\phi_{d,e}$. Therefore, because (1) extensible interior loop contributions are only considered after contributions that lead to an overall asymptotic upper bound of $O(N^2)$, (2) base pairs bounding extensible loops are not considered more than once, and (3) there are a total of $O(N^2)$ possible base pairs bounding extensible loops in a sequence of length N, the overall sampling algorithm scales as $O(N^2)$. This matches the asymptotic scaling of Ding and Lawrence[15], while including the complete class of large interior loops, some of which they exclude.

A.6.6 Suboptimal structure generation

In many cases, the core features of a complex ensemble may be summarized by its MFE proxy structure(s) (Equation 2.10), $s_{\mathrm{MFE}'}$, or the set of all stacking states below a given free energy gap $\overline{\Gamma}_{\mathrm{subopt}}(\phi, \Delta G_{\mathrm{gap}})$ (Equation 2.11). The set $\overline{\Gamma}_{\mathrm{subopt}}(\phi, \Delta G_{\mathrm{gap}})$ can be equivalently viewed as the set of structures corresponding to stacking states $s^{\shortparallel}$ whose equilibrium probability $\overline{p}(\phi, s^{\shortparallel})$ is at least $\overline{p}_{\mathrm{gap}} \equiv \exp(-(\overline{\Delta G}(\phi, s^{\shortparallel}_{\mathrm{MFE}}) + \Delta G_{\mathrm{gap}})/kT)(\overline{Q}(\phi))^{-1}$. $\overline{\Gamma}_{\mathrm{subopt}}(\phi, \Delta G_{\mathrm{gap}})$ is necessarily just a set of $s_{\mathrm{MFE}'}$ when $\Delta G_{\mathrm{gap}} = 0$, and algorithmically we therefore focus on calculation of $\overline{\Gamma}_{\mathrm{subopt}}(\phi, \Delta G_{\mathrm{gap}})$.

The program flow for determining suboptimal structures is controlled by a stack data structure containing partial structures $\{s\}$. Each partial structure s represents all structures consistent with a given set of elements that have energies below an energy gap. It is defined as a tuple of 1) a priority queue of recursion elements, 2) a free energy and 3) a list of base pairs.

Structure generation proceeds by popping the highest priority element λ from the top partial structure s on the stack. The appropriate recursion for the element is used to iterate through the set of all alternate conditional ensemble contributions. For each alternate contribution falling below the given energy gap, a new partial structure s' is generated from s. If a given contribution contained no elements and the priority queue of s' is empty, s' is output as a complete structure; otherwise, s' is pushed on the stack. The algorithm begins by pushing a partial structure corresponding to $Q_{1,N}$ onto the stack and proceeds until the stack is empty.

Using a stack data structure, the algorithm runs in a depth first manner to discover completed structures as early as possible. This allows emitting completed structures in a streaming fashion while additional structures are determined. Here we give an algorithm which yields $\overline{\Gamma}_{\mathrm{subopt}}(\phi, \Delta G_{\mathrm{gap}})$ from sequence ϕ of length $N \equiv |\phi|$:

1. Initialize empty stack $\mathcal{S}$ of partial structures and empty multiset L of complete structures.
2. Create parent partial structure s containing just the element $\lambda = (\varnothing, 1, N)$ and push it onto $\mathcal{S}$.
3. While $\mathcal{S}$ is not empty:
    a) Pop the first partial structure s off of the stack $\mathcal{S}$.
    b) If there are no elements in s, it is complete, so add it to L and continue the while loop.
    c) Otherwise, dequeue the first element λ from s.
    d) Update the free energy of s via $s_{\mathrm{energy}} \leftarrow s_{\mathrm{energy}} - Q_{\textsc{MinSum}}(\lambda)$.
    e) If λ denotes any element $Q^b_{i,j}$, add a base pair $i \cdot j$ in structure s.
    f) Calculate the generator $G = R_{\textsc{ArgGap}}(\lambda)(\kappa)$. For exposition, this may be achieved with the coroutine $\kappa(x, \Lambda)$ which yields (x, Λ) and returns κ. In practice, the loop below was programmatically implemented via a callback function.
    g) For each contribution (x, Λ) in G where $s_{\mathrm{energy}} + x \leq \Delta G_{\mathrm{gap}} + \overline{\Delta G}(\phi, s^{\shortparallel}_{\mathrm{MFE}})$:

   i. Initialize a new partial structure s' from s by copying the priority queue and list of base pairs from s and energy $s'_{\text{energy}} = s_{\text{energy}} + x$.
   ii. For each element $\lambda' \in \Lambda$, enqueue λ' into the priority queue of partial structure s'.
   iii. Push s' onto the stack $\mathcal{S}$.

4. Return L.

The resultant algorithm complexity is within $O(|L|N^2)$. This bound reflects the worst-case of a set of L structures which contain no common recursion elements. Each structure must then be independently backtracked, incurring the worst-case $O(N^2)$ complexity bound of Section A.6.5. Because the number of structures returned $|L|$ is sequence-dependent and potentially exponential in N, we did not attempt to bound the complexity further.

A.7 Distinguishability Issues

For a complex of L strands, the ensemble $\overline{\Gamma}$ treats each strand as distinct while the ensemble Γ treats strands with the same sequence as indistinguishable. Both ensembles have conceptual utility as they provide different perspectives when examining the physical properties of a complex. In laboratory experiments, strands with the same sequence are typically indistinguishable, so calculations over ensemble Γ are crucial for comparison to experimental data (e.g., equilibrium secondary structure probabilities and equilibrium complex concentrations). On the other hand, calculations over ensemble $\overline{\Gamma}$ can sometimes provide information that is valuable precisely because it cannot be measured experimentally (e.g., equilibrium base-pairing probability matrix).

All of the dynamic programs described in the present work operate on ensemble $\overline{\Gamma}$ using free energy model (2.1) where each strand is treated as distinct. This is a matter of algorithmic necessity, as the free energy model (2.4) used for ensemble Γ contains a symmetry correction that depends on the global rotational symmetry R of each secondary structure $s \in \Gamma$. For efficiency reasons, the dynamic programs avoid explicitly enumerating each structure, instead operating on local loop free energies to incorporating information for multiple structures simultaneously while operating only on local loop free energies. As a result, the dynamic programs cannot incorporate a different global rotational symmetry correction for each structure because they never have access to global structural information. However, to facilitate comparisons to experimental data, physical quantities calculated using a dynamic program over ensemble $\overline{\Gamma}$ using physical model (2.1) can be post-processed to obtain the corresponding physical quantities over ensemble Γ using physical model (2.4). In the following sections, we outline the situation for each physical quantity treated in the present work.

A.7.1 Partition function

The partition function dynamic program calculates $\overline{Q}(\phi) = Q_{1,N}$ (for a complex with N nucleotides) over ensemble $\overline{\Gamma}$ using free energy model (2.1) treating all strands as distinct. The Distinguishability Correction Theorem of Dirks et al.[2] shows that this quantity can be used to calculate the partition function $Q(\phi)$ over ensemble Γ using physical model (2.4) treating strands with the same sequence as indistinguishable. For convenience, we include the associated definitions and proof[2] here to enable extension of this analysis to other physical quantities.

Consider a complex of L strands with ordering π, where some of the strands may be indistinguishable. Let $\mathcal{G}$ be the group of $v(\pi)$ cyclic permutations mapping each strand to a strand of the same species. For example, $v(\pi) = 4$ for $\pi = AAAA$, $v(\pi) = 3$ for $\pi = ABABAB$, and $v(\pi) = 2$ for $\pi = ABAABA$, $v(\pi) = 1$ for $\pi =$ AAB, where the elements of $\mathcal{G}$ correspond to all rotations of a polymer graph that map strands of type $A \longrightarrow A$ and strands of type $B \longrightarrow B$. We term $v(\pi)$ the periodic strand repeat of the complex with ordering π.

For complexes in which all strands are distinct, $v(\pi) = 1$. Complexes containing multiple copies of the same strand species may have $v(\pi) > 1$, in which case the calculated partition function will be incorrect for ensemble Γ and free energy model (2.4) due to symmetry and redundancy errors

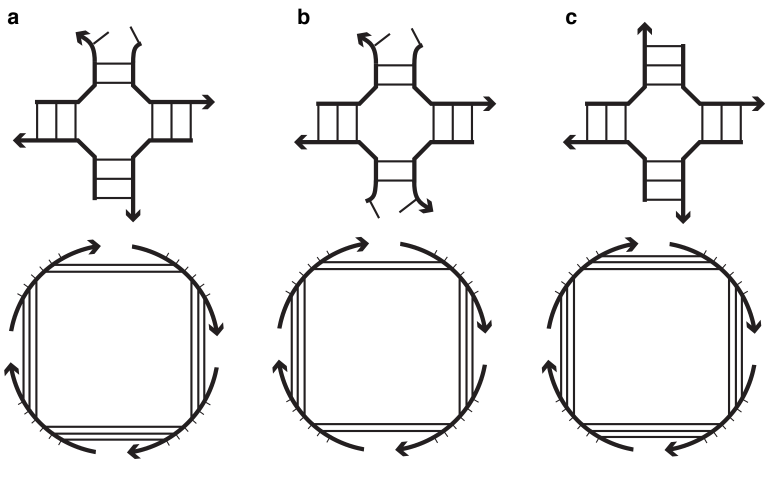


Figure A.33: Example secondary structures and polymer graphs for a complex with strand ordering π =AAAA. The four strands have the same sequence and are distinct in ensemble $\overline{\Gamma}$ (each with a unique identifier in {1,2,3,4}) but indistinguishable in ensemble Γ. The partition function dynamic program operates on ensemble $\overline{\Gamma}$. After completing a calculation, if the strand identifiers are conceptually removed with the goal of converting the partition function $\overline{Q}(\phi)$ from ensemble $\overline{\Gamma}$ to the partition function $Q(\phi)$ in ensemble Γ, different structures in $\overline{\Gamma}$ have different rotational symmetries and different redundancies in Γ. Structures with an R-fold rotational symmetry are missing a penalty of $+kT \log R$ to the free energy model and hence are overweighted in the partition function by a factor of R. Structures with an S-fold redundancy are overcounted in the partition function by a factor of S. (a) 1-fold (i.e., no) rotational symmetry; 4-fold redundancy (4 indistinguishable structures as each stem plays the role of having 2 base pairs). (b) 2-fold rotational symmetry; 2-fold redundancy (2 indistinguishable structures as each opposing pair of stems plays the role of having 2 base pairs). (c) 4-fold rotational symmetry; 1-fold (i.e., no) redundancy.

that are different for different structures in the ensemble. For example, consider a complex with strand ordering $\pi = AAAA$ (Figure A.33), that contains structures with either a 1-fold (i.e., no symmetry), 2-fold, or 4-fold rotational symmetry. Each of these cases will be treated incorrectly from the perspective of ensemble Γ and physical model (2.4). Dirks et al.[2] show that the symmetry and redundancy errors interact in such a way that they can be exactly and simultaneously corrected.

Consider an arbitrary secondary structure $s \in \overline{\Gamma}$. A permutation $g \in \mathcal{G}$ acts on a secondary structure s by relabeling strand identifiers: $g(s) = \{i_{g(m)} \cdot j_{g(n)} : i_m \cdot j_n \in s\}$. The stabilizer of s, $\mathcal{G}_s = \{g \in \mathcal{G} : g(s) = s\}$, is the set of cyclic permutations of strand identifiers (rotations of the polymer graph) that map s onto itself. The order of the rotational symmetry of the physical complex with secondary structure s is given by $|\mathcal{G}_s|$, requiring a correction of $+kT \log |\mathcal{G}_s|$ to the standard loop-based free energy.

The orbit of s in $\mathcal{G}$, $\mathcal{G}(s) = \{g(s) \in \overline{\Gamma} : g \in \mathcal{G}\}$, is the subset of $\overline{\Gamma}$ corresponding to the images of s under the permutations of the group $\mathcal{G}$. Note that the members of $\mathcal{G}(s)$ represent secondary structures within $\overline{\Gamma}$ that would be indistinguishable if the unique identifiers were removed from strands of the same species. Consequently, the partition function contribution of secondary structure $s \in \Gamma$ will be overcounted by a factor of $|\mathcal{G}(s)|$ because the dynamic program treats elements of the orbit as algorithmically distinct even though they are physically indistinguishable.

The orbit-stabilizer theorem of group theory[16] provides the useful relationship

$$|\mathcal{G}_s||\mathcal{G}(s)| = |\mathcal{G}| = v(\pi), \ \forall\, s \in \overline{\Gamma}$$

linking the symmetry and redundancy effects. Crucially, the product $|\mathcal{G}_s||\mathcal{G}(s)|$ depends only on the strand ordering π and is independent of the specific secondary structure $s \in \overline{\Gamma}$.

Theorem 1 (Partition Function Distinguishability Correction) *For a complex with strand ordering π, if the partition function dynamic program yields $\overline{Q}(\phi)$ for ensemble $\overline{\Gamma}$, then the partition function for ensemble Γ accounting for both symmetry and redundancy corrections is $Q(\phi) = \overline{Q}(\phi)/v(\pi)$.*

Proof. The partition function algorithm applied to ensemble $\overline{\Gamma}$ yields

$$\overline{Q}(\phi) = \sum_{s\in\overline{\Gamma}} \exp\{-\overline{\Delta G}(\phi, s)/kT\}. \tag{A.82}$$

The partition function for ensemble Γ is then

$$\begin{aligned} Q(\phi) &= \sum_{s\,\in\Gamma} \exp\{-\Delta G(\phi, s)/kT\} \\ &= \sum_{s\,\in\Gamma} \exp\{-(\overline{\Delta G}(\phi, s) + kT\ \log|\mathcal{G}_s|)/kT\} && \text{(A.83)} \\ &= \sum_{s\,\in\Gamma}\sum_{\sigma\in\mathcal{G}(s)} \frac{1}{|\mathcal{G}(\sigma)|} \exp\{-(\overline{\Delta G}(\phi, \sigma) + kT\ \log|\mathcal{G}_\sigma|)/kT\} && \text{(A.84)} \\ &= \sum_{s\,\in\overline{\Gamma}} \frac{1}{|\mathcal{G}(s)|} \exp\{-(\overline{\Delta G}(\phi, s) + kT\ \log|\mathcal{G}_s|)/kT\} && \text{(A.85)} \\ &= \frac{1}{v(\pi)} \sum_{s\,\in\overline{\Gamma}} \exp\{-\overline{\Delta G}(\phi, s)/kT\} && \text{(A.86)} \\ &= \frac{\overline{Q}(\phi)}{v(\pi)}. && \text{(A.87)} \end{aligned}$$

Thus, the symmetry and redundancy corrections combine to give a uniform factor $v(\pi)^{-1}$ that is independent of the structure $s \in \overline{\Gamma}$, enabling exact conversion of $\overline{Q}(\phi)$ into $Q(\phi)$. □

The partition function $Q(\phi)$ for ensemble Γ is suitable for calculating physical quantities that will be compared to experimental measurements in which strands of the same species are indistinguishable (e.g., equilibrium secondary structure probabilities $p(\phi, s)$ or equilibrium complex concentrations x). The corresponding complex free energy is

$$\Delta G(\phi) = -kT \log Q(\phi), \tag{A.88}$$

which should not be confused with $\Delta G(\phi, s)$, the free energy of a single secondary structure $s \in \Gamma$.

A.7.2 Equilibrium secondary structure probability

In ensemble $\overline{\Gamma}$ treating all strands as distinct, the equilibrium probability of any secondary structure $s \in \overline{\Gamma}$ is:

$$\overline{p}(\phi, s) = \frac{1}{\overline{Q}(\phi)} \exp\{-\overline{\Delta G}(\phi, s)/kT\} \tag{A.89}$$

where $\overline{Q}(\phi)$ is the partition function over ensemble $\overline{\Gamma}$ treating all strands as distinct and $\overline{\Delta G}(\phi, s)$ is calculated using (2.1).

In ensemble Γ treating strands with the same sequence as indistinguishable, the equilibrium probability of any secondary structure $s \in \Gamma$ is:

$$p(\phi, s) = \frac{1}{Q(\phi)} \exp\{-\Delta G(\phi, s)/kT\} \tag{A.90}$$

where $Q(\phi)$ is calculated using (A.87) and $\Delta G(\phi, s)$ is calculated using (2.4).

The relationship between the probabilities in the two ensembles is given by:

$$\begin{aligned}
p(\phi, s) &= \frac{1}{Q(\phi)} \exp\{-\Delta G(\phi, s)/kT\} && \text{(A.91)}\\
&= \frac{v(\pi)}{\overline{Q}(\phi)} \exp\{-[\overline{\Delta G}(\phi, s) + kT \log |\mathcal{G}_s|]/kT\} && \text{(A.92)}\\
&= \frac{v(\pi)}{\overline{Q}(\phi)} \sum_{\sigma \in \mathcal{G}(s)} \frac{1}{|\mathcal{G}(\sigma)|} \exp\{-[\overline{\Delta G}(\phi, \sigma) + kT \log |\mathcal{G}_\sigma|]/kT\} && \text{(A.93)}\\
&= \frac{1}{\overline{Q}(\phi)} \sum_{\sigma \in \mathcal{G}(s)} \exp\{-\overline{\Delta G}(\phi, \sigma)/kT\} && \text{(A.94)}\\
&= \sum_{\sigma \in \mathcal{G}(s)} \overline{p}(\phi, \sigma) && \text{(A.95)}\\
& && \text{(A.96)}
\end{aligned}$$

where the structures in the set $\mathcal{G}(s)$ for $s \in \overline{\Gamma}$ become redundant if the distinct identifiers are removed from strands of the same species. Hence, $p(\phi, s)$ is the sum of the (identical) probabilities $\overline{p}(\phi, s)$ of these redundant structures.

A.7.3 Equilibrium base-pairing probabilities

Using a bactrack-free dynamic program, the matrix of equilibrium base-pairing probabilities $P(\phi)$ is calculated over ensemble $\overline{\Gamma}$ using free energy model (2.1) treating all strands as distinct. One may visualize a thought experiment in which all strands, all nucleotides, and all base-pairs are distinct, with equilibrium base-pairing probabilities available for each of these distinct base pairs. The probabilities in this matrix are not directly comparable to experimental measurements in which strands of the same sequence are indistinguishable, but nonetheless provide a valuable and detailed window into the complex ensemble.

Let

$$p(i_1 \cdot j_2) \tag{A.97}$$

denote the equilibrium probability for base-pair $(i_1 \cdot j_2)$ with nucleotide i of a strand with identifier 1 pairing to nucleotide j of a strand with identifier 2. Let

$$p(i_1) \tag{A.98}$$

denote the equilibrium probability that base i of a strand with identifier 1 is unpaired.

A.7.4 Structure sampling

The simultaneous sampling algorithm Boltzmann samples a set of J secondary structures

$$\overline{\Gamma}_{\text{sample}}(\phi, J) \tag{A.99}$$

from ensemble $\overline{\Gamma}$ using free energy model (2.1) treating all strands as distinct. Unlike the equilibrium base-pairing probability matrix $P(\phi)$, by averaging or clustering the sampled structures, it is possible to examine correlations between base pairs. As the number of sampled structures increases, the average structural properties over the sampled set recover the equilibrium base-pairing probability matrix:

$$P(\phi) = \lim_{J \to \infty} \frac{1}{J} \sum_{s \in \overline{\Gamma}_{\text{sample}}} S(s) \tag{A.100}$$

A set of J structures $\overline{\Gamma}_{\text{sample}}(\phi, J)$ sampled from ensemble $\overline{\Gamma}$ where all strands are distinct can be post-processed to generate a set of structures $\Gamma_{\text{sample}}(\phi, J)$ sampled from ensemble Γ where strands with the same sequence are indistinguishable.

For ensemble $\overline{\Gamma}$ with free energy model (2.1), a structure $s \in \overline{\Gamma}$ is Boltzmann sampled with probability $\overline{p}(\phi, s)$ by the sampling dynamic program, yielding an integer number of samples $\overline{n}_{\text{sample}}(\phi, s) \in \{0, \ldots, J\}$. Conceptually, for ensemble Γ with free energy model (2.4), a structure $s \in \Gamma$ would be Boltzmann sampled with probability $p(\phi, s)$. We have previously derived the relationship (A.95) between the equilibrium probabilities in the two ensembles:

$$p(\phi, s) = \sum_{\sigma \in \mathcal{G}(s)} \overline{p}(\phi, \sigma) \tag{A.101}$$

The equilibrium probability of a structure $s \in \Gamma$ is simply the sum of the equilibrium probabilities of the structures $\sigma \in \mathcal{G}(s)$ that are indistinguishable in ensemble $\overline{\Gamma}$ upon removal of their unique identifiers. Hence, the sample count for Boltzmann sampling from ensemble Γ with free energy model (2.1) is obtained by summing the sample counts for the structures $\sigma \in \mathcal{G}(s)$ that are indistinguishable in ensemble $\overline{\Gamma}$ upon removal of their unique identifiers:

$$n_{\text{sample}}(\phi, s) = \sum_{\sigma \in \mathcal{G}(s)} \overline{n}_{\text{sample}}(\phi, \sigma). \tag{A.102}$$

A.7.5 Equilibrium complex concentrations

Consider a test tube ensemble containing a set of strand species Ψ^0 interacting to form the set of complex species Ψ. To calculate the set of equilibrium concentrations $x_\Psi \equiv x_c \ \forall c \in \Psi$, we first calculate the set of partition functions Q_Ψ using (A.87). The complex concentrations x_Ψ are then the unique solution to the strictly convex optimization problem[2]:

$$\min_{x_\Psi} \sum_{c \in \Psi} x_c(\log x_c - \log Q_c - 1) \tag{A.103a}$$

$$\text{subject to} \quad \sum_{c \in \Psi} A_{i,c} x_c = x_i^0 \quad \forall i \in \Psi^0. \tag{A.103b}$$

Here, A is the stoichiometry matrix such that $A_{i,c}$ is the number of strands of type i in complex c and x_i^0 denotes the total concentration of strand species i in the test tube. Following Dirks et al.,[2], this problem is solved efficiently in the dual form as an unconstrained convex optimization problem using a trust-region method with a Newton dog-leg step[17] using Cholesky decomposition for the Newton matrix inversions.

A.7.6 Ensemble pair fractions

If a complex contains some indistinguishable strands, distinguishability effects arise at the secondary structure level in the form of rotational symmetry corrections and algorithmic overcounting corrections (Section A.7.1). New distinguishability issues arise when examining individual base pairs within these secondary structures[2]. For example, consider a complex $\pi = AAB$ involving two indistinguishable copies of strand A (with identifiers 1 and 2) and one copy of strand B (with identifier 3). Periodic strand repeat $v(\pi) = 1$ so no symmetry and overcounting corrections are required for any structure $s \in \Gamma$. However, base pairs $(i_1 \cdot j_3)$ and $(i_2 \cdot j_3)$ are indistinguishable since strands 1 and 2 are both of type A. Likewise, without the global structural context, the inter- and intra-strand base pairs $(i_1 \cdot j_2)$ and $(i_1 \cdot j_1)$ are also indistinguishable. Fortunately, the equilibrium base-pairing probabilities calculated over ensemble Γ (Section A.7.3) can be used to calculate base-pairing observables that account for this indistinguishability.

First, consider a complex in which strands with the same sequence are indistinguishable. Let Θ be the set of strand species in the complex and $\{\theta\}$ be the set of all strand identifiers corresponding to strands of type $\theta \in \Theta$ (hence $L = \sum_{\theta \in \Theta} |\{\theta\}|$). We define the expected number of base pairs between base i on strands of type $A \in \Theta$ and base j on strands of type $B \in \Theta$ to be $E(i_{\{A\}} \cdot j_{\{B\}}) \in [0, \min(|\{A\}|, |\{B\}|)]$. For a given complex,

$$E(i_{\{A\}} \cdot j_{\{B\}}) = \sum_{l_A \in \{A\}} \sum_{l_B \in \{B\}} p(i_{l_A} \cdot j_{l_B})$$

represents a sum over the contributions of each type of distinct base pair, where each term $p(i_{l_A} \cdot j_{l_B})$ is an equilibrium base-pairing probability (A.97).

Now consider a test tube in which strands with the same sequence are indistinguishable. Let Ψ^0 denote the set of strand species that interact to form the set of complex species Ψ. For a complex

$k \in \Psi$, let $E_k(i_{\{A\}} \cdot j_{\{B\}})$ denote the expectation value that base i of strand species $A \in \Theta_k$ pairs to base j of strand species $B \in \Theta_k$, where $\Theta_k \subseteq \Psi^0$ denotes the set of strand species that appear in complex k. For a test tube ensemble at equilibrium, the expected concentration of base pairs between base i of strands of type A and base j of strands of type B is

$$x(i_A \cdot j_B) = \sum_{k \in \Psi} x_k E_k(i_{\{A\}} \cdot j_{\{B\}}).$$

For experimental studies, it is usually more convenient to measure the expected fraction of A strands or B strands that form this base pair:

$$f_A(i_A \cdot j_B) = x(i_A \cdot j_B)/x_A^0 \tag{A.104}$$

$$f_B(i_A \cdot j_B) = x(i_A \cdot j_B)/x_B^0, \tag{A.105}$$

respectively. These ensemble pair fractions are conceptually suitable for comparison to a FRET experiment designed to measure formation of a base-pair between base i of strands of type A with base j of strands of type B.

Similarly, the concentration $x(i_A)$ of strand species $A \in \Psi^0$ with base i unpaired is

$$x(i_A) = 1 - \sum_{B \in \Psi^0} \sum_{j=1}^{N_B} x(i_A \cdot j_B),$$

and the fraction of A strands that have base i unpaired is

$$f_A(i_A) = x(i_A)/x_A^0. \tag{A.106}$$

The total concentration of unpaired bases in solution is

$$x_{\text{unpaired}} = \sum_{A \in \Psi^0} x(i_A) = \sum_{k \in \Psi} x_k \sum_{j=1}^{N_k} P^{j,j}(\phi_k) \tag{A.107}$$

and the total fraction of unpaired bases in solution is

$$f_{\text{unpaired}} = x_{\text{unpaired}} / \sum_{A \in \Psi^0} x_A^0 N_A \tag{A.108}$$

The total fraction unpaired is conceptually suitable for comparison to an absorbance measurement.

A.7.7 MFE free energy and secondary structure

The MFE dynamic program returns the free energy of the MFE stacking state in ensemble $\overline{\Gamma}$ using free energy model (2.1):

$$\overline{\Delta G}(\phi, s^{\shortparallel}_{\text{MFE}}). \tag{A.109}$$

Note that the MFE algorithm does not return the free energy of the MFE secondary structure s_{MFE} but rather the free energy of the MFE stacking state $s^{\shortparallel}_{\text{MFE}}$. This is a consequence of the recursions operating on stacking state as the elementary state. The backtracking dynamic program then returns the secondary structure

$$s_{\mathrm{MFE}'} = \{s \in \overline{\Gamma} | s^{\shortparallel}_{\mathrm{MFE}} \in s, s^{\shortparallel}_{\mathrm{MFE}}(\phi) = \arg\min_{s^{\shortparallel} \in \overline{\Gamma}^{\shortparallel}} \overline{\Delta G}(\phi, s^{\shortparallel})\}. \tag{A.110}$$

that contains $s^{\shortparallel}_{\mathrm{MFE}}$ within its subensemble. Note that this is not the MFE secondary structure, s_{MFE}, but rather a proxy $s_{\mathrm{MFE}'}$ that contains $s^{\shortparallel}_{\mathrm{MFE}}$ within its subensemble. This is a consequence of ??. The free energy of this secondary structure can be evaluated in ensemble $\overline{\Gamma}$ using (2.1) to yield $\overline{\Delta G}(\phi, s_{\mathrm{MFE}'})$ or in ensemble Γ using (2.4) to yield $\Delta G(\phi, s_{\mathrm{MFE}'})$.

Because the recursions operate on stacking states as the elementary state, it is not clear how to calculate the MFE free energy $\overline{\Delta G}(\phi, s_{\mathrm{MFE}})$ and secondary structure s_{MFE} for ensemble $\overline{\Gamma}$. As a result, there is also no starting point for post-processing these results to calculate $\Delta G(\phi, s_{\mathrm{MFE}})$ or s_{MFE} for ensemble Γ.

This situation is not entirely satisfactory. By definition, an MFE secondary structure has the highest equilibrium probability, $\overline{p}(\phi, s_{\mathrm{MFE}})$, in structural ensemble $\overline{\Gamma}$. However, $\overline{p}(\phi, s_{\mathrm{MFE}})$ can nonetheless be arbitrarily small due to competition from other structures in $\overline{\Gamma}$. For ensembles where $p(\phi, s_{\mathrm{MFE}})$ is non-negligible, an attractive alternative to the deterministic approach is to use Boltzmann sampling to discover the MFE secondary structure. One advantage of the random approach is that it determines MFE status based on secondary structure s rather than subensemble stacking state $s^{\shortparallel} \in s$.

Sampling is performed for ensemble $\overline{\Gamma}$ treating all strands as distinct. Suppose that the identity of s_{MFE} is unknown, as is its free energy $\overline{\Delta G}(\phi, s_{\mathrm{MFE}})$ and its equilibrium probability $\overline{p}(\phi, s_{\mathrm{MFE}})$. The probability, p_{fail}, that a sample of J structures does not include a structure s_{MFE} that has probability $\overline{p}(\phi, s_{\mathrm{MFE}}) \geq p_{\mathrm{min}}$ is

$$p_{\mathrm{fail}} \leq (1 - p_{\mathrm{min}})^J. \tag{A.111}$$

Inverting this relationship, for a given p_{min}, we can calculate the number of samples, J, required to assure a failure probability no higher than p_{fail}:

$$J \geq \frac{\log p_{\mathrm{fail}}}{\log(1 - p_{\mathrm{min}})} \approx \frac{\log p_{\mathrm{fail}}}{-p_{\mathrm{min}}} \tag{A.112}$$

Because the dependence of J on p_{fail} is logarithmic, it is inexpensive to reduce p_{fail} for fixed p_{min}. For example, for $p_{\mathrm{min}} = 0.01$, we have $J \geq 688$ for $p_{\mathrm{fail}} = 10^{-3}$ and $J \geq 2750$ for $p_{\mathrm{fail}} = 10^{-12}$. However, the required number of samples is sensitive to the value of p_{min} (the assumed lower bound in the MFE probability). For example, holding $p_{\mathrm{fail}} = 10^{-12}$ fixed, we require $J \geq 27,618$ samples for $p_{\mathrm{min}} = 0.001$ and $J \geq 276,297$ samples for $p_{\mathrm{min}} = 0.0001$. While that number of samples remains affordable using the new simultaneous sampling algorithm (Figure 2.14), if the MFE probability becomes vanishingly small, the required number of samples would grow too large to be practical. On the other hand, if the MFE probability is vanishingly small, the MFE structure may not provide a useful summary of the equilibrium base-pairing properties of the ensemble (in which case the equilibrium base-pairing probability matrix $P(\phi)$ will continue to provide such a summary).

After sampling J structures using the new simultaneous sampling method, let $p_{\mathrm{MFE}'}$ denote the highest probability of the sampled structures. The probability that there exists an (undiscovered) MFE structure with $\overline{p}_{\mathrm{MFE}} \geq \overline{p}_{\mathrm{MFE}'}$ is bounded by

$$p_{\mathrm{fail}} \leq [1 - \overline{p}(\phi, s_{\mathrm{MFE}'})]^J. \tag{A.113}$$

Hence, after sampling J structures, it is straightforward calculate the probability that the true MFE structure was not identified. If desired, additional samples can be performed to increase J (potentially identifying a higher $p_{\mathrm{MFE}'}$) and further reduce the failure rate.

One of the drawbacks of the deterministic approach of (2.9) and (2.10) is that it does not treat ensemble Γ where strands with the same sequence are indistinguishable, which is the circumstance for typical experimental measurements. However, the random MFE algorithm can be applied using samples from ensemble Γ, in which case the a posteriori failure bound (A.113) is replaced by

$$p_{\mathrm{fail}} \leq [1 - p(\phi, s_{\mathrm{MFE}'})]^J. \tag{A.114}$$

The MFE free energy $\Delta G(\phi, s_{\mathrm{MFE}'})$ is then directly comparable to the complex free energy $\Delta G(\phi)$ (A.88) for ensemble Γ with

$$\Delta G(\phi) \leq \Delta G(\phi, s_{\mathrm{MFE}'}). \tag{A.115}$$

See Section A.8.6 for an empirical comparison of deterministic and random algorithms in calculating the MFE free energy and secondary structure.

A.8 Additional studies

A number of empirical studies of algorithm outputs and efficiency are presented in this section. All such studies, as in the main text, were done using default NUPACK parameters (RNA sequences at 37 °C, 1 M Na^+, 0 M Mg^{2+}, allowing wobble (GU) pairs). Unless otherwise noted, the recursions with full coaxial and dangle stacking were used. All computations were run on AWS EC2 C5 instances in serial.

A.8.1 Empirical dependence of ensemble size on complex size

We performed calculations to measure the number of secondary structures, $|\overline{\Gamma}|$, and stacking states, $|\overline{\Gamma}^{\parallel}|$, for a set of randomly generated complexes. The results demonstrate empirically that both $|\overline{\Gamma}|$ and $|\overline{\Gamma}^{\parallel}|$ grow exponentially with the number of nucleotides. Least-squares linear regressions were performed on the log-linear data and are plotted as lines below. The fits were $|\overline{\Gamma}| = 0.00156 \cdot 1.770^N$ ($r = 0.9999994$) and $|\overline{\Gamma}^{\parallel}| = 0.00650 \cdot 2.023^N$ ($r = 0.9999996$). These results are sequence-dependent and are not generalizable across different classes of complexes.

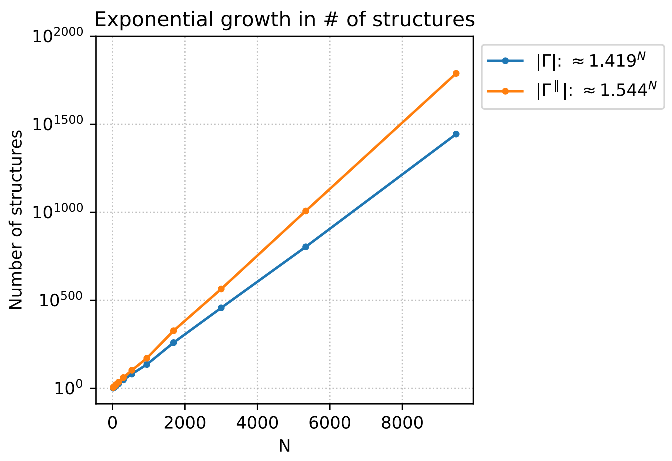


Figure A.34: Calculated ensemble sizes for random sequences (3 RNA strands of equal length).

A.8.2 Empirical dependence of partition function on complex size

We measured the partition functions at default parameters for both random complexes and engineered duplexes to benchmark when overflow is predicted to occur using traditional approaches in varying floating point formats. The duplexes are stabilized and overflow at smaller complex sizes than the random complexes, which generally have higher free energies. These results are sequence-dependent and are not generalizable across different classes of complexes. For example, without the overflow-safe evaluation algebra, edge-case sequences such as the repeating sequence GGG...CCC... have been observed to cause overflow at sequence sizes as low as 4,500 nt in quadruple precision.

For the designed duplexes, NUPACK design was used to design helices composed of two equal length strands constrained to be reverse complements of each other. The normalized complex ensemble defect[18] was reduced below 1%. Linear regressions of $\log Q$ gave fits of $\log Q = 0.515N - 7.31$

(r = 0.999986) for the random sequences and $\log Q = 1.56N - 1.68$ (r = 0.999993) for the duplexes. Respectively for single, double, and quadruple precision, the random complexes are predicted to overflow at 187 nt, 1,393 nt, and 22,080 nt, while the designed duplexes are predicted to overflow at 58 nt, 456 nt, and 7,275 nt.

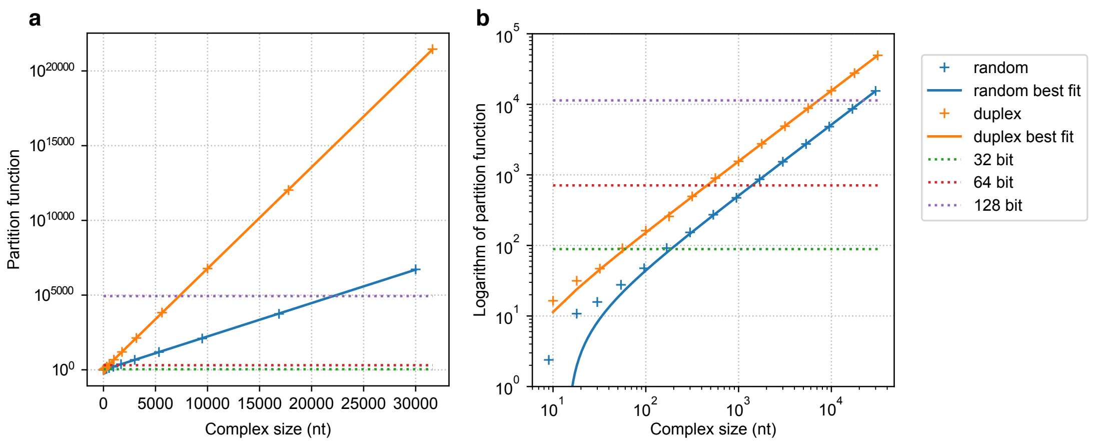


Figure A.35: (a) Partition functions for random complexes and engineered duplexes using the coaxial stacking recursions (reproduction of Figure 2.10(c)). (b) Same data, plotted in terms of the logarithm of the partition function. Equally sized complexes of 3 RNA strands were generated for the random complexes. Means across 5 replicates are plotted. The thresholds for overflow to occur using different precisions are also plotted, demonstrating that the overflow-safe algebra allows for calculations of larger complex sizes.

A.8.3 Relative cost of partition function, pair probability, and MFE calculations

We also benchmarked the relative performance of the $O(N^3)$ analysis calculations for partition function, pair probabilities, and minimum free energy (MFE). The MFE algorithm is the fastest algorithm in general. The partition function algorithm is slightly slower, especially at larger complex sizes as overflow-safe computation must be used. The pair probabilities algorithm is roughly 2× the cost of the partition function algorithm, except for a spike to ≈ 3× at complex sizes around the threshold for when overflow-safe computation starts to be used.

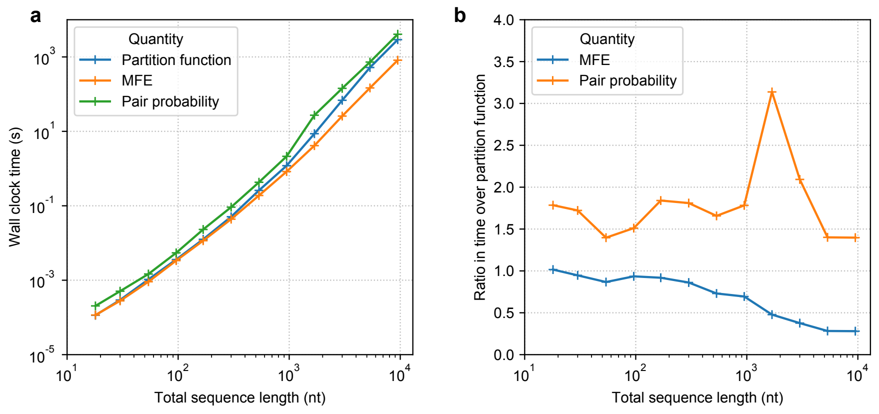


Figure A.36: (a) Partition function, MFE and pair probabilities algorithm costs. (b) Ratio in cost over partition function. Results are for complexes of 3 equally sized random RNA sequences. Each algorithm has an $O(N^3)$ complexity. Means across 5 replicates are plotted.

A.8.4 Robustness and speed of partition function calculations with different data types

We compared the costs of performing traditional partition function computation in single precision, double precision, and overflow-safe single precision computation. The production algorithm is also shown, which dynamically switches from single precision to double precision to the overflow-safe evaluation algebra in single precision as is required by the sequence considered. We see that the overflow-safe evaluation algebra is around 2× slower than the others, but it enables calculation of larger complexes.

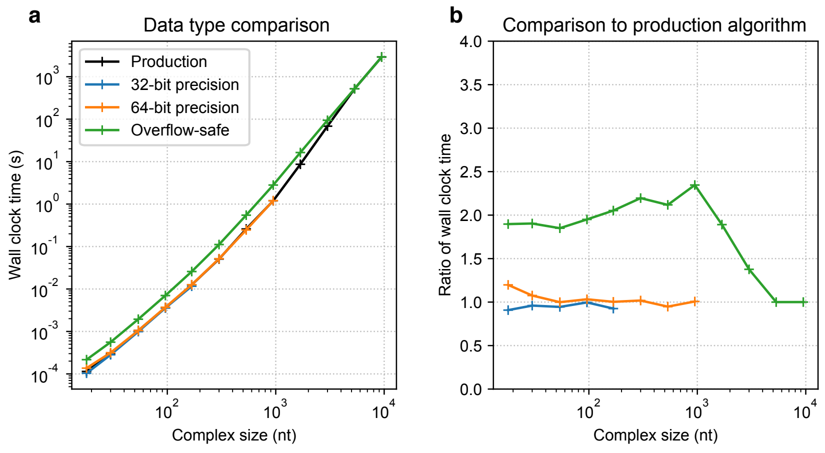


Figure A.37: (a) Comparison of partition function computation times vs the underlying precision used. Using the non-overflow safe evaluation algebra necessitates larger floating point formats for longer sequences as shown in Section A.8.2 (b) Ratios of computation time divided by that for the production algorithm. The same sequences were used as in Section A.8.3. Means across 5 replicates are plotted.

A.8.5 Performance of simultaneous vs serial structure sampling

Structure sampling for random complexes. We studied the performance of sampling algorithms on random complexes, which are expected to favor the simultaneous algorithm the least due to the lack of stabilized structures with low free energy. For each replicate, three equally sized random RNA sequences were generated, ranging in length from 10 to 3000 nucleotides each, and between 10^1 and 10^6 samples were taken per sequence. Five replicates were used for each sequence size.

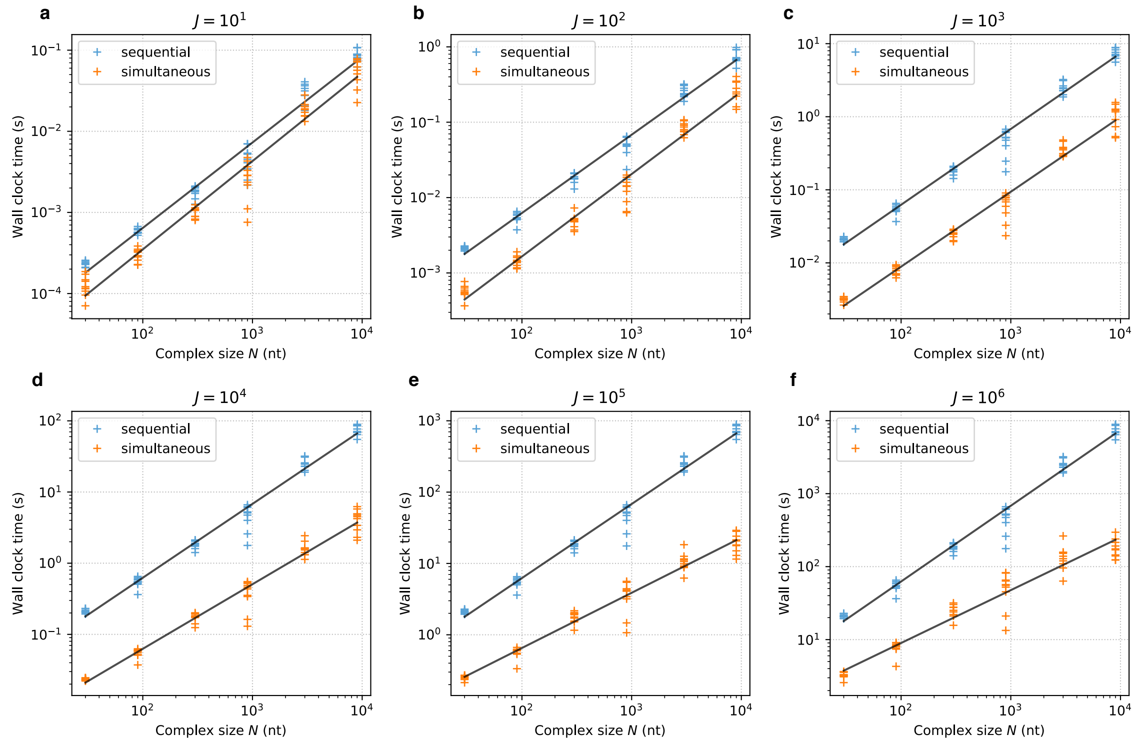


Figure A.38: (a-f) Sampling computation times for each individual complex considered. Univariate regressions (Section A.8.5) for each J are also plotted.

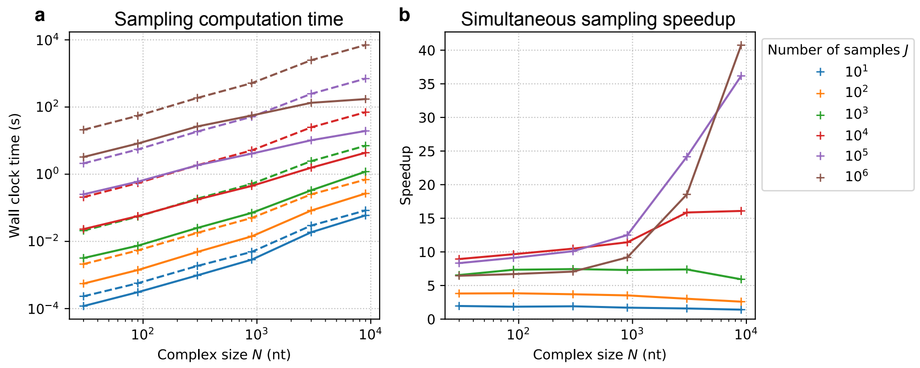


Figure A.39: Mean sampling computation times for all replicates of equal N, plotted with respect to N. (a) Comparison of simultaneous (solid) and sequential (dashed) algorithms. (b) Speedup of simultaneous algorithm.

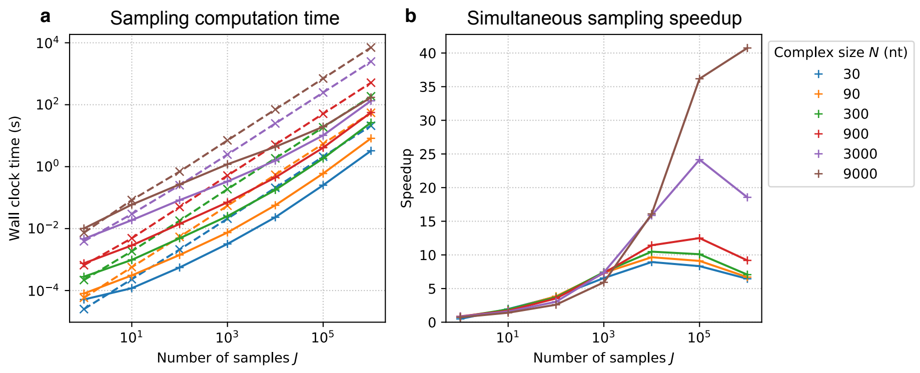


Figure A.40: Mean sampling computation times for all replicates of equal N, plotted with respect to J. (a) Comparison of simultaneous (solid) and sequential (dashed) algorithms. (b) Speedup of simultaneous algorithm.

Structure sampling for designed complexes. We studied the performance of sampling algorithms on designed complexes, which are expected to favor the simultaneous algorithm more than random complexes. Sequences used ranged in length from 10 to 3000 nucleotides, and between 10^1 and 10^6 samples were taken per complex. For each sequence length, designed complexes consisted of 3 strands of that sequence length. Each complex was designed in two steps. First, the MFE structure was computed for a set of 3 random strands of the same length as the complex to design. Second, this MFE structure was used as the target structure for sequence design via complex ensemble defect minimization[18]. Complexes were designed to have a normalized complex ensemble defect less that 1%. For each sequence length, 10 unique 3-stranded complexes were designed and then sampled from.

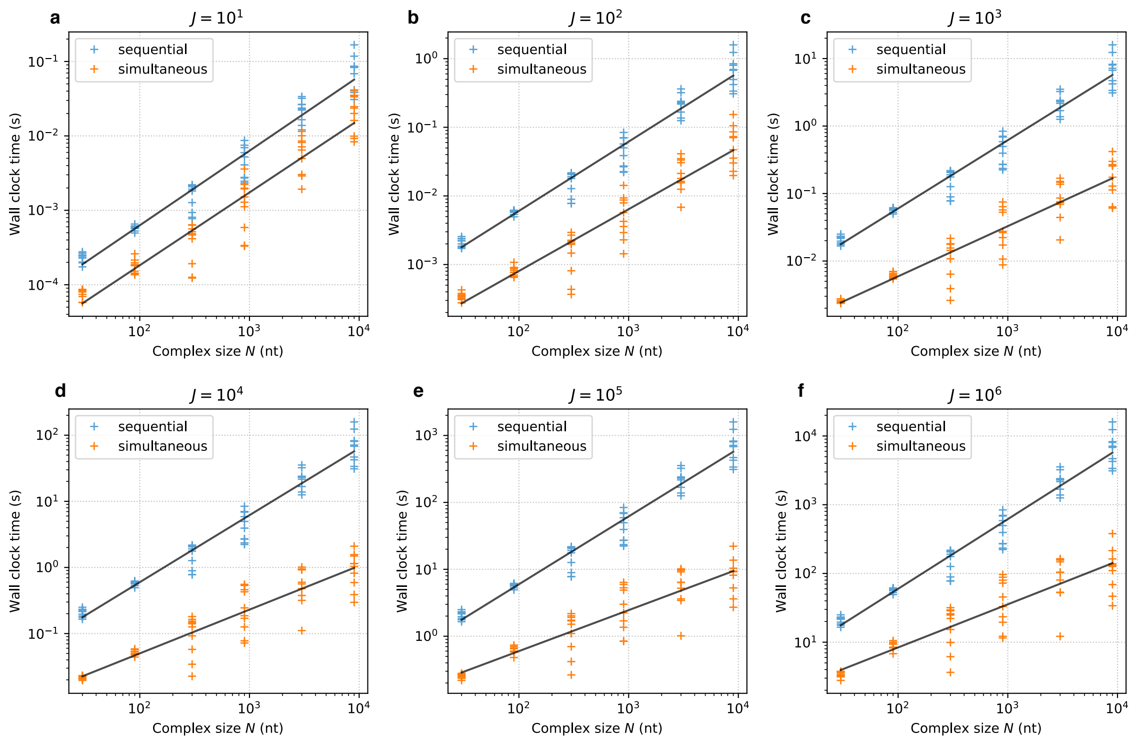


Figure A.41: (a-f) Sampling computation times for each individual complex considered. Univariate regressions (Section A.8.5) for each J are also plotted.

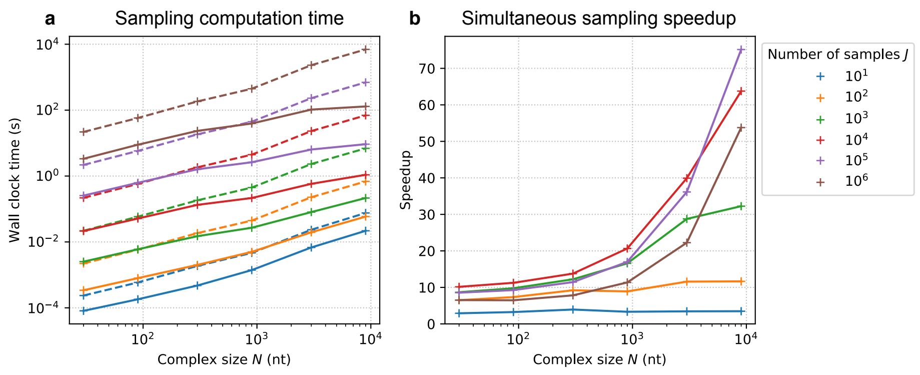


Figure A.42: Mean sampling computation times for all replicates of equal N, plotted with respect to N. (a) Comparison of simultaneous (solid) and sequential (dashed) algorithms. (b) Speedup of simultaneous algorithm.

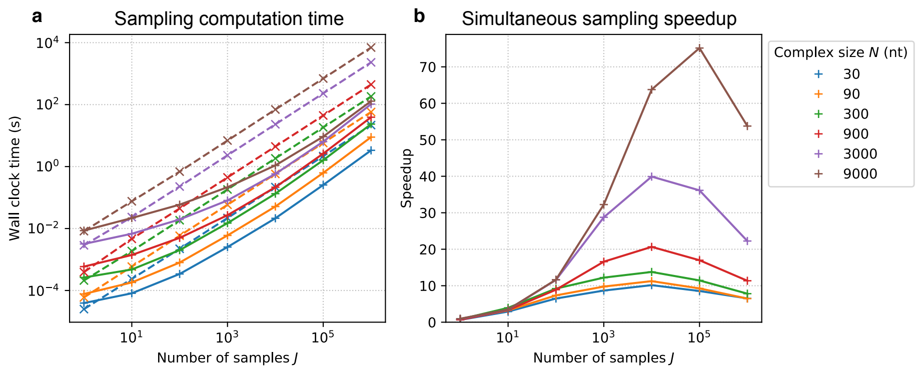


Figure A.43: Mean sampling computation times for all replicates of equal N, plotted with respect to J. (a) Comparison of simultaneous (solid) and sequential (dashed) algorithms. (b) Speedup of simultaneous algorithm. At extremely high sample numbers J, the cost of sorting begins to be observed, lessening the speedup of the simultaneous approach.

Empirical complexity estimates. In this section we report empirical measures of the complexities of the sequential and simultaneous Boltzmann sampling algorithms over the random and designed complexes from Sections A.8.5 and A.8.5.

Sequences	Method	Complexity in N	Complexity in J	Prefactor (s)	r-value
Designed	Sequential	1.011	0.994	6.003-07	0.996
Designed	Simultaneous	0.776	0.791	1.339-06	0.973
Random	Sequential	1.040	0.992	5.570-07	0.998
Random	Simultaneous	0.934	0.773	1.014-06	0.987

Table A.1: Bivariate least-squares linear regression of the fit $\log T \approx \alpha_N \log N + \alpha_J \log J + \log P$ such that $T \approx P N^{\alpha_N} J^{\alpha_J}$, with T the computation time, α_N the complexity in N, α_J the complexity in J, and P the prefactor. For sequential sampling, α_J is extremely close to 1 because the calculation is simply a repetition of a single sample J times.

Complexes	Method	N	Complexity in J	Prefactor (s)	r-value
Designed	Sequential	30	0.988	0.000023	0.999
Designed	Simultaneous	30	0.827	0.000016	0.976
Designed	Sequential	90	0.997	0.000058	0.999
Designed	Simultaneous	90	0.860	0.000028	0.988
Designed	Sequential	300	0.995	0.000153	0.995
Designed	Simultaneous	300	0.832	0.000068	0.968
Designed	Sequential	900	1.002	0.000412	0.994
Designed	Simultaneous	900	0.811	0.000210	0.968
Designed	Sequential	3000	0.991	0.002341	0.997
Designed	Simultaneous	3000	0.738	0.001123	0.964
Designed	Sequential	9000	0.991	0.007035	0.992
Designed	Simultaneous	9000	0.678	0.003485	0.961
Random	Sequential	30	0.988	0.000023	0.999
Random	Simultaneous	30	0.791	0.000025	0.978
Random	Sequential	90	1.002	0.000053	0.999
Random	Simultaneous	90	0.830	0.000041	0.992
Random	Sequential	300	0.993	0.000194	0.999
Random	Simultaneous	300	0.817	0.000149	0.990
Random	Sequential	900	0.990	0.000486	0.996
Random	Simultaneous	900	0.796	0.000386	0.984
Random	Sequential	3000	0.984	0.002962	0.997
Random	Simultaneous	3000	0.723	0.003144	0.990
Random	Sequential	9000	0.994	0.007752	0.998
Random	Simultaneous	9000	0.681	0.009725	0.991

Table A.2: Univariate least-squares linear regressions of the fit $\log T \approx \alpha_J \log J + \log P$ for fixed values of N such that $T \approx PJ^{\alpha_J}$, with T the computation time, α_J the complexity in J, and P the prefactor. For sequential sampling, α_J is extremely close to 1 because the calculation is simply a repetition of a single sample J times.

Complexes	Method	J	Complexity in N	Prefactor (s)	r-value
Designed	Sequential	10	1.001	0.000006	0.978
Designed	Simultaneous	10	0.978	0.000002	0.955
Designed	Sequential	100	1.009	0.000058	0.979
Designed	Simultaneous	100	0.900	0.000013	0.949
Designed	Sequential	1000	1.012	0.000567	0.980
Designed	Simultaneous	1000	0.743	0.000193	0.934
Designed	Sequential	10000	1.012	0.005639	0.980
Designed	Simultaneous	10000	0.662	0.002377	0.919
Designed	Sequential	100000	1.013	0.056201	0.980
Designed	Simultaneous	100000	0.612	0.035748	0.903
Designed	Sequential	1000000	1.013	0.562028	0.980
Designed	Simultaneous	1000000	0.627	0.466401	0.893
Random	Sequential	10	1.053	0.000005	0.988
Random	Simultaneous	10	1.088	0.000002	0.982
Random	Sequential	100	1.038	0.000052	0.990
Random	Simultaneous	100	1.093	0.000011	0.987
Random	Sequential	1000	1.037	0.000525	0.991
Random	Simultaneous	1000	1.024	0.000079	0.987
Random	Sequential	10000	1.037	0.005242	0.991
Random	Simultaneous	10000	0.907	0.000960	0.986
Random	Sequential	100000	1.038	0.052318	0.991
Random	Simultaneous	100000	0.773	0.018521	0.980
Random	Sequential	1000000	1.037	0.524730	0.991
Random	Simultaneous	1000000	0.724	0.319781	0.967

Table A.3: Univariate least-squares linear regressions of the fit $\log T \approx \alpha_N \log N + \log P$ for fixed values of J such that $T \approx PN^{\alpha_N}$, with T the computation time, α_N the complexity in N, and P the prefactor.

A.8.6 Different approaches for approximating the MFE structure

In this section we report empirical comparisons between different approaches for determining the minimum free energy secondary structure in a complex ensemble using the coaxial stacking recursions. In the simplest approach, we ran the suboptimal structure algorithm to determine the minimum free energy stacking state and its associated secondary structure. We also investigated using the Boltzmann sampling algorithm by taking the lowest free energy secondary structure from a set of 100,000 samples. These benchmarks were run using RNA, default parameters, and sequences designed for structures from the multistranded engineered test set from Reference 19.

For small complexes, we observe that the sampling approach sometimes yields slightly more stable secondary structures. For large complexes, the suboptimal structure approach usually yields a more stable structure. Each approach produces relatively similar secondary structures in terms of base pairing. Overall, the secondary structure corresponding to the minimum free energy stacking state seems to be a reasonable approximation of the minimum free energy secondary structure.

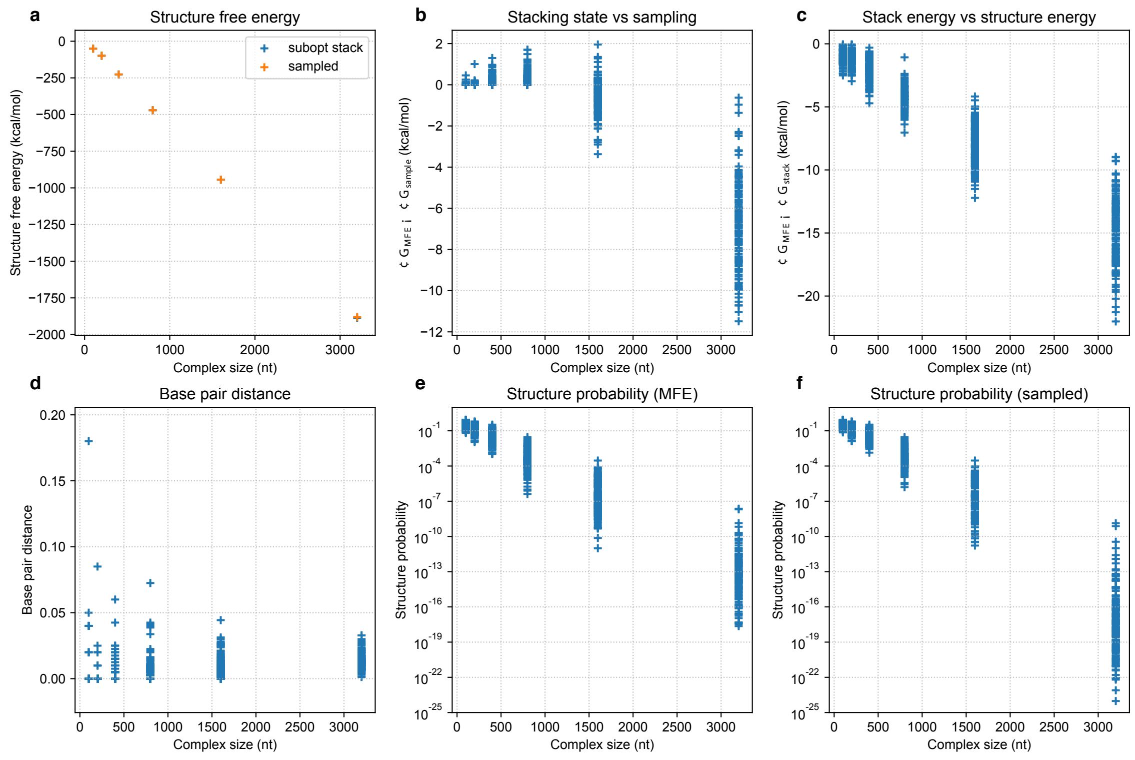


Figure A.44: (a) Comparison of mean structure free energies discovered by the MFE algorithm and the sampling algorithm (respectively $\Delta G(\phi, s_{\mathrm{MFE}'})$ and $\min_{s \in \Gamma_{\mathrm{sample}}(\phi, 10^5)} \Delta G(\phi, s)$). Markers are visually coincident. (b) Residuals from plot (a). (c) Comparison of structure free energy $\Delta G(\phi, s_{\mathrm{MFE}'})$ and stacking state free energy $\Delta G(\phi, s^{||}_{\mathrm{MFE}})$ discovered by the MFE algorithm. (d) Fraction of differently paired nucleotides between the MFE algorithm and the sampling algorithm. (e) Equilibrium probability of the secondary structure discovered by the MFE algorithm. (f) Equilibrium probability of the secondary structure discovered by the sampling algorithm.

A.9 Validation test suite

Listed below is a subset of unit and regression tests used to validate the implementation of these algorithms. Over 100 unit tests were run via continuous integration on a dedicated JetBrains TeamCity server, comprising $O(10^7)$ test case assertions in total. Since thermodynamic algorithms were implemented generically with respect to model recursions, we focused on partition function and structure count to verify correctness for different model recursions.

A.9.1 Exhaustive enumeration algorithms

Enumeration of complex ensemble without coaxial and dangle stacking subensembles. For validation purposes, we provide pseudocode to enumerate all possible secondary structures for a given complex ensemble (strand ordering) in a recursive manner below. This implementation is chosen for its simplicity (rather than efficiency), and relies on imposing a total ordering on base pair indices (i, j) via the function CompareBasePair. EnumerateSecondaryStructures is a generator function which yields all possible secondary structures by delegating to the inner generator function EnumerateHigherStructures.

EnumerateSecondaryStructures yields all possible secondary structures, which may include disconnected complexes. For a single-complex ensemble, any disconnected structures may be easily removed in post-processing.

```
CompareBasePair($p, p'$)
    $i, j \leftarrow p$
    $i', j' \leftarrow p'$
    return $i < i'$ or ($i = i'$ and $j < j'$)

EnumerateSecondaryStructures($\phi$)
    $N \leftarrow$ Length($\phi$)
    $s \leftarrow$ UnpairedStructure($N$)
    $p \leftarrow (0, 0)$
    EnumerateHigherStructures($\phi, s, p$)

EnumerateHigherStructures($\phi, s, p$)
    $N \leftarrow$ Length($\phi$)
    for $i \in [1 : N]$
        for $j \in [i + 1 : N]$
            $p' \leftarrow (i, j)$
            if CanPair($\phi, i, j$) and CompareBasePair($p, p'$)
                $s' \leftarrow$ AddBasePair($s, p'$)
                EnumerateHigherStructures($\phi, s', p'$)
    Yield $s$
```

Algorithm A.8: Enumeration of secondary structures consistent with sequence ϕ

Enumeration of coaxial and dangle stacking subensemble for a single secondary structure. Stacking states are constructed hierarchically from a given secondary structure by first finding all coaxial stacking states (without dangles) for each loop and then finding all dangle stacking states consistent with each coaxial stacking state. The top-level function EnumerateStackingStates-ForStructure is a generator function that which yields all possible stacking states for a given secondary structure and sequence. It does this by delegating to the function EnumerateLoop-StackingStates, which yields all stacking states for a given loop within the secondary structure. This relies on the function GetValidMasks which returns a list of all possible coaxial stacking states within the loop (neglecting dangles) and the function GetLoopStackingStates, which yields all stacking states for a loop consistent with a given coaxial stacking state.

The function Product takes a list of generators (or lists), G, and returns a tuple of elements, one from each generator (or list). This lazily generates all tuples from the cartesian product of the sets generated by each generator in G. This is equivalent to a nested for loop with $|G|$ levels of nesting.

The function REGIONS splits the sequences of the loop into subsequences between the base pairs and nicks. For example, in a multiloop with base pairs $i \cdot j$, $d \cdot e$, and $f \cdot g$ with $i < d < e < f < g < j$, the function returns the list of sequences $[\phi_{[i:d]}, \phi_{[e:f]}, \phi_{[g:j]}]$. In an exterior loop with a on the $3'$ side of the nick and b on the $5'$ side of the nick and base pairs $i \cdot j$ and $d \cdot e$ with $a < i < j < d < e < b$, the function returns the list of sequences $[\phi_{[a:i]}, \phi_{[j:d]}, \phi_{[e:b]}]$. For each region of the loop, its stacking state is indicated with a number: 0 for no nucleotides stacking, 1 for a coaxial stack between the two adjacent base pairs, 3 for the $3'$−most nucleotide stacking on the $3'$ base pair, 5 for the $5'$−most nucleotide stacking on the $5'$ base pair, and 8 for the $3'$−most nucleotide stacking on the $3'$ base pair and the $5'$−most nucleotide stacking on the $5'$ base pair (if these are distinct nucleotides).

The functions LEFT and RIGHT return true if the regions to the left or right of the given region are involved in a coaxial stacking interaction and false if not. This prevents dangle stacking on a base pair already in a coaxial stack. In exterior loops, if a is the nucleotide that is on the $3'$ side of the nick and b is the nucleotide that is on the $5'$ side of the nick, then LEFT returns true for the region containing a and RIGHT returns true for the region containing b. This prevents a and b from erroneously being included in invalid dangle states. The function LEFTNICK returns true if there is a nick $5'$ to the region. The function RIGHTNICK returns true if there is a nick $3'$ to the region. The function ATNICK returns true if either LEFTNICK or RIGHTNICK returns true.

The function BINARYVECTOR(number, width) produces a vector of 1s and 0s that is the binary representation of the input number with zero-padding up to the input width.

The function TYPE returns the type of loop, i.e. “hairpin”, “stack”, “bulge”, “interior”, “multi” or “exterior”. The function NOSTACKING returns an object indicating that this loop does not having fine-grained stacking states.

```
EnumerateStackingStatesForStructure($\phi$, $s$)
    $G \leftarrow []$
    for $l \in$ Loops($s$)
        Append($G$, EnumerateLoopStackingStates($\phi$, $l$))
    for $[\omega] \in$ Product($G$)
        $s'' \leftarrow$ StackingState($s$, $[\omega]$)
        Yield($s''$)
```

Algorithm A.9: Enumeration of stacking states for a given structure s and sequence ϕ

```
EnumerateLoopStackingStates($\phi, l$)
    $\phi^R \leftarrow$ Regions($\phi, l$)
    $V^{\text{mask}} \leftarrow$ GetValidMasks($\phi^R, l$)
    if $V^{\text{mask}} = []$
        Yield NoStacking()
    for $v^{\text{mask}} \in V^{\text{mask}}$
        GetLoopStackingStates($\phi^R, v^{\text{mask}}$)
```

Algorithm A.10: Enumeration of stacking state for a given loop l in sequence ϕ

```
GetValidMasks($\phi^R, l$)
    if ¬(Type($l$) = “multi” or Type($l$) = “exterior”)
        return []
    $v^{\text{indices}} \leftarrow []$
    for $i \in [1 : |\phi^R|]$
        if $|\phi_i^R| = 2$
            Append($v^{\text{indices}}, i$)
    $V^{\text{mask}} \leftarrow []$
    for $i \in [0 : 2^{|v^{\text{indices}}|}]$
        $t^{\text{mask}} \leftarrow$ BinaryVector($i, |v^{\text{indices}}|$)
        $v^{\text{mask}} \leftarrow []$
        for $i \in [1 : |\phi^R|]$
            if $i \in v^{\text{indices}}$
                Append($v^{\text{mask}}, t_i^{\text{mask}}$)
            else
                Append($v^{\text{mask}}, 0$)
        $c \leftarrow$ true
        for $i \in [1 : |\phi^R|]$
            if $v_i^{\text{mask}} = 1$ and ((Right($\phi_i^R, v^{\text{mask}}, l$) and ¬(RightNick($\phi_i^R, v^{\text{mask}}, l$)) or
            (Left($\phi_i^R, v^{\text{mask}}, l$) and ¬(LeftNick($\phi_i^R, v^{\text{mask}}, l$)))
                $c \leftarrow$ false
        if $c =$ true
            Append($V^{\text{mask}}, v^{\text{mask}}$)
    return $V^{\text{mask}}$
```

Algorithm A.11: Enumeration of coaxial stacking states for given loop l containing sequence regions ϕ^R

```
GetLoopStackingStates($\phi^R, l, v^{\text{mask}}$)
    $G^R \leftarrow []$
    for $i \in |\phi^R|$
        if $|\phi^R_i| \geq 3$ or $(|\phi^R_i| \geq 2$ and AtNick$(\phi^R_i))$
            if Right($\phi^R_i, v^{\text{mask}}, l$) and $\neg$Left($\phi^R_i, v^{\text{mask}}, l$)
                Append($G^R, [0, 5]$)
            elseif Left($\phi^R_i, v^{\text{mask}}, l$) and $\neg$Right($\phi^R_i, v^{\text{mask}}, l$)
                Append($G^R, [0, 3]$)
            elseif $\neg$(Left($\phi^R_i, v^{\text{mask}}, l$) or Right($\phi^R_i, v^{\text{mask}}, l$))
                if $|\phi^R_i| = 3$
                    Append($G^R, [0, 3, 5]$)
                elseif $|\phi^R_i| > 3$
                    Append($G^R, [0, 3, 5, 8]$)
        else
            Append($G^R, [v^{\text{mask}}_i]$)
    for $[x] \in$ Product($G^R$)
        $\omega \leftarrow$ LoopStackingState($l, [x]$)
        Yield($\omega$)
```

Algorithm A.12: Enumeration of stacking states for given loop l containing sequence regions ϕ^R with fixed coaxial stacking state v^{mask}

Enumeration of complex ensemble with coaxial and dangle stacking subensembles. To obtain all the stacking states for the complex, the above functions EnumerateSecondaryStructures and EnumerateStackingStatesForStructure are composed.

```
EnumerateStackingStates($\phi$)
    for $s \leftarrow$ EnumerateSecondaryStructures($\phi$)
        EnumerateStackingStatesForStructure($\phi, s$)
```

Algorithm A.13: Enumeration of stacking states consistent with sequence ϕ

A.9.2 Unit tests

Individual loop free energies. Test loop free energies for all loop types vs manual calculations, including contributions from dangles and coaxial stacking.

Secondary structure enumeration. Check structure counts, partition functions, and MFEs vs enumerated calculation for the $O(N^3)$ algorithm, with wobble pairs on and off, on RNA and DNA, with each possible dangle setting (“none", “some", “all", “coax").

Partition functions and counts with coaxial stacking. Check that coaxial stacking algorithm matches vectorized and unvectorized reference implementations based on literal translation of the pseudocode for partition function and count, single and multiple strands.

Overflow-safe evaluation algebra. Check that partition functions agree with non-overflow variants for each dangle type (“none", “some", “all", “coax"), for single and multiple strand complexes, for 500 random sequences of RNA and DNA.

Consistency between all possible data types. Verify that all results are equal for partition function calculations on complexes up to 4 strands, using both $O(N^4)$ and $O(N^3)$ algorithms, for (1) 32 bit non-overflow, 32 bit overflow, (2) 32 bit non-overflow, 64 bit overflow, (3) 64 bit non-overflow, 32 bit overflow, and (4) 32 bit non-overflow, 64 bit non-overflow, 32 bit overflow data types.

Consistency when using caching methodology for multistranded calculations. Verify consistency of caching used in multistranded algorithm for pair probability and partition function, $O(N^3)$ and $O(N^4)$ algorithms, caching on and off, different orders of evaluations of requested complexes, on random sequences and edge cases we found during development.

Boltzmann sampled structure generation. Check that structures are generated at around the right probability in the limit of a large number of samples. Check that the sample pair probability matrix approaches that given by the dynamic program as the number of samples is increased.

Comparisons of different complexity algorithms. Check that $O(N^3)$ and $O(N^4)$ structure counts and partition functions agree for each dangle type (“none", “some", “all", “coax"), for single strand and 3-strand complexes, for 500 sets of random RNA sequences.

A.9.3 Regression tests

Individual secondary structure free energies. Test structure free energies vs NUPACK 3 for RNA and DNA, single and multiple strand complexes, wobble pairs on and off, random temperatures, and each dangle stacking heuristic that was possible in NUPACK 3 (“none", “some", “all").

Necklace generation. Test example necklace generation up to ($|\Psi_0| = 10$, $L_{max} = 4$) vs NUPACK 3. Check that the number of free energies returned via dynamic programs is equal to the number of necklaces requested.

Partition functions and counts compared to NUPACK 3. Check that structure counts and partition functions agree with NUPACK 3, for each dangle type (“none", “some", “all"), for wobble pairs on and off, for single and multiple strand complexes, for random sequences of RNA and DNA.

Comparison with NUPACK 3 for different parameter sets. Check that structure counts and partition functions agree with NUPACK 3 for random temperatures, random choices of parameter set, random concentrations of Na^+ and Mg^{++}, for single and multiple strands of RNA and DNA.

Minimum free energy structures. Check agreement with NUPACK 3 with wobble pairs on and off, single and multiple strands, for random RNA and DNA sequences.

Pair probability matrices. Check matrices vs. NUPACK 3 for RNA and DNA, single and multiple strands, wobble pairs on and off, random sequences and historical edge case sequences.

Suboptimal and MFE structures. Check that generated structures are identical to NUPACK 3 for DNA and RNA, 0 and 0.4 kcal/mol energy gaps, wobble pairs on and off, single and multiple strands, random sequences and historical edge cases.

Equilibrium concentrations. Check convergence solution accuracy vs implementation in NUPACK 3 for concentration solver for free energies of random RNA complexes, free energies of random DNA complexes, and free energies of isolated edge cases which we found to not converge well in earlier versions of the code.

Bibliography

[1] I. Tinoco Jr., O. Uhlenbeck, and M. Levine. “Estimation of Secondary Structure in Ribonucleic Acids”. In: *Nature* 230 (1971), pp. 362–367.

[2] R. M. Dirks, J. S. Bois, J. M. Schaeffer, E. Winfree, and N. A. Pierce. “Thermodynamic Analysis of Interacting Nucleic Acid Strands”. In: *SIAM Rev.* 49.1 (2007), pp. 65–88.

[3] M. J. Serra and D. H. Turner. “Predicting Thermodynamic Properties of RNA”. In: *Methods Enzymol.* 259 (1995), pp. 242–261.

[4] T. Xia, J. SantaLucia Jr., M. Burkard, R. Kierzek, S. Schroeder, X. Jiao, C. Cox, and D. Turner. “Thermodynamic Parameters for an Expanded Nearest-Neighbor Model for Formation of RNA Duplexes with Watson-Crick Base Pairs”. In: *Biochemistry* 37.42 (1998), pp. 14719–14735.

[5] D. H. Mathews, J. Sabina, M. Zuker, and D. H. Turner. “Expanded Sequence Dependence of Thermodynamic Parameters Improves Prediction of RNA Secondary Structure”. In: *J. Mol. Biol.* 288 (1999), pp. 911–940.

[6] M. Zuker. “Mfold Web Server for Nucleic Acid Folding and Hybridization Prediction”. In: *Nucleic Acids Res.* 31.13 (2003), pp. 3406–3415.

[7] Z. J. Lu, D. H. Turner, and D. H. Mathews. “A Set of Nearest Neighbor Parameters for Predicting the Enthalpy Change of RNA Secondary Structure Formation”. In: *Nucleic Acids Res.* 34.17 (2006), pp. 4912–4924. ISSN: 0305-1048. DOI: 10.1093/nar/gkl472.

[8] D. H. Turner and D. H. Mathews. “NNDB: the nearest neighbor parameter database for predicting stability of nucleic acid secondary structure”. In: *Nucleic Acids Res.* 38 (2010), pp. D280–D282.

[9] J. SantaLucia Jr. “A Unified View of Polymer, Dumbbell, and Oligonucleotide DNA Nearest-Neighbor Thermodynamics”. In: *Proc. Natl. Acad. Sci. U. S. A.* 95.4 (1998), pp. 1460–1465.

[10] N. Peyret. “Prediction of Nucleic Acid Hybridization: Parameters and Algorithms”. Thesis. 2000.

[11] S. Bommarito, N. Peyret, and J. SantaLucia. “Thermodynamic Parameters for DNA Sequences with Dangling Ends”. In: *Nucleic Acids Res.* 28.9 (2000), pp. 1929–1934. ISSN: 0305-1048. DOI: DOI10.1093/nar/28.9.1929.

[12] J. SantaLucia Jr. and D. Hicks. “The Thermodynamics of DNA Structural Motifs”. In: *Annu. Rev. Biophys. Biomol. Struct.* 33 (2004), pp. 415–440. ISSN: 1056-8700.

[13] R. T. Koehler and N. Peyret. “Thermodynamic Properties of DNA Sequences: Characteristic Values for the Human Genome”. In: *Bioinformatics* 21.16 (2005), pp. 3333–3339. ISSN: 1367-4803. DOI: Doi10.1093/Bioinformatics/Bti530.

[14] R. M. Dirks and N. A. Pierce. “A Partition Function Algorithm for Nucleic Acid Secondary Structure Including Pseudoknots”. In: *J. Comput. Chem.* 24 (2003), pp. 1664–1677.

[15] Y. Ding and C. Lawrence. “A Statistical Sampling Algorithm for RNA Secondary Structure Prediction”. In: *Nucleic Acids Res.* 31.24 (2003), pp. 7280–7301.

[16] J. Gallian. *Contemporary Abstract Algebra*. New York: Houghton Mifflin, 2002.

[17] J. Nocedal and S. Wright. *Numerical Optimization*. New York: Springer, 1999.

[18] B. R. Wolfe, N. J. Porubsky, J. N. Zadeh, R. M. Dirks, and N. A. Pierce. “Constrained Multistate Sequence Design for Nucleic Acid Reaction Pathway Engineering”. In: *J. Am. Chem. Soc.* 139 (2017), pp. 3134–3144.

[19] J. N. Zadeh, B. R. Wolfe, and N. A. Pierce. “Nucleic Acid Sequence Design via Efficient Ensemble Defect Optimization”. In: *J. Comput. Chem.* 32 (2011), pp. 439–452. doi: 10.1002/jcc.21633.

Appendix B

Constrained Multistate Sequence Design for Nucleic Acid Reaction Pathway Engineering: Supplementary Information

This appendix was adapted from material in the supplementary info of B. R. Wolfe*, N. J. Porubsky*, J. N. Zadeh, R. M. Dirks, and N. A. Pierce. "Constrained Multistate Sequence Design for Nucleic Acid Reaction Pathway Engineering". In: *Journal of the American Chemical Society* 139.8 (2017), pp. 3134–3144. ISSN: 15205126. DOI: 10.1021/jacs.6b12693.

B.1 Algorithm

The constrained multistate test tube design algorithm described in the present work builds on the test tube design algorithm described by Wolfe and Pierce[1]. The two algorithms were developed concurrently so that the notation and concepts employed for sequence design over the ensemble of a single test tube would generalize naturally to performing sequence design over the ensemble of an arbitrary number of test tubes subject to user-specified sequence constraints. Readers interested in a detailed understanding of the present algorithm will benefit from reading the Algorithm section of Reference 1, which contains thorough descriptions of a subset of the algorithmic ingredients used in the present work. For conciseness, if an algorithmic ingredient requires no or minimal generalization and there is little risk of confusion, we simply refer to Reference 1 for details (using the same section heading for clarity). If some generalization in notation or concept is required and there is a risk of confusion, we restate the description of Reference 1 with updated details below (again using the same section heading for clarity). If a new algorithmic ingredient is required in the present setting, we provide full details below.

B.1.1 Secondary Structure Model

The secondary structure, s, of one or more interacting nucleic acid strands is defined by a set of base pairs[2]. A *polymer graph* representation of a secondary structure is constructed by ordering the strands around a circle, drawing the backbones in succession from $5'$ to $3'$ around the circumference with a *nick* between each strand, and drawing straight lines connecting paired bases. A secondary structure is *unpseudoknotted* if there exists a strand ordering for which the polymer graph has no crossing lines. A secondary structure is *connected* if no subset of the strands is free of the others. A *complex* of interacting strands is specified as a strand ordering, π, corresponding to the *structural ensemble*, Γ, containing all connected polymer graphs with no crossing lines.[1]. (We dispense with our prior convention[2–4] of calling this entity an *ordered complex*.) See Section S1.3 of Reference 1 for a discussion of distinguishability issues. A *test tube* may contain an arbitrary number of strand species interacting to form an arbitrary number of complex species in a dilute solution.

The sequence, ϕ, of a complex is specified as a list of bases $\phi^a \in \{\mathtt{A},\mathtt{C},\mathtt{G},\mathtt{U}\}$ for $a = 1,\ldots,|\phi|$ (T replaces U for DNA). Each base pair in a secondary structure is a Watson–Crick pair (A·U or C·G) or a wobble pair (G·U). For sequence ϕ and secondary structure $s \in \Gamma$, the *free energy*, $\Delta G(\phi, s)$, is calculated using nearest-neighbor empirical parameters for RNA[5–7] in 1M Na^+ or for DNA supercitesantalucia98,zuker03 in user-specified concentrations of Na^+ and Mg^{++}[8,9]. These physical models have practical utility for the analysis[10–18] and design[10,11,19–48] of functional nucleic acid systems, and provide the basis for rational analysis and design of equilibrium base-pairing in test tube ensembles for reaction pathway engineering.

B.1.2 Analyzing Equilibrium Base-Pairing in the Multistate Test Tube Ensemble

Let Ψ_h^0 denote the set of strand species that interact in test tube $h \in \Omega$ to form the set of complex species Ψ_h. For complex $j \in \Psi_h$, with sequence ϕ_j and structural ensemble Γ_j, the partition function

$$Q(\phi_j) = \sum_{s \in \Gamma_j} \exp\left[-\Delta G(\phi_j, s)/k_B T\right]$$

can be used to calculate the equilibrium probability of any secondary structure $s \in \Gamma_j$:

$$p(\phi_j, s) = \exp\left[-\Delta G(\phi_j, s)/k_B T\right]/Q(\phi_j).$$

Here, k_B is the Boltzmann constant and T is temperature. The equilibrium base-pairing properties of complex j are characterized by the base-pairing probability matrix $P(\phi_j)$, with entries $P^{a,b}(\phi_j) \in [0,1]$ corresponding to the probability,

$$P^{a,b}(\phi_j) = \sum_{s \in \Gamma_j} p(\phi_j, s) S^{a,b}(s),$$

that base pair $a \cdot b$ forms at equilibrium within ensemble Γ_j. Here, $S(s)$ is a *structure matrix* with entries $S^{a,b}(s) = 1$ if structure s contains base pair $a \cdot b$ and $S^{a,b}(s) = 0$ otherwise. For convenience, the structure and probability matrices are augmented with an extra column to describe unpaired bases. The entry $S^{a,|s|+1}(s)$ is unity if base a is unpaired in structure s and zero otherwise; the entry $P^{a,|\phi_j|+1}(\phi_j) \in [0,1]$ denotes the equilibrium probability that base a is unpaired over ensemble Γ_j. Hence the row sums of the augmented $S(s)$ and $P(\phi_j)$ matrices are unity.

Let $Q_{\Psi_h} \equiv Q_j \;\; \forall j \in \Psi_h$ denote the set of partition functions for the complexes in tube h. The set of equilibrium concentrations, x_{h,Ψ_h}, (specified as mole fractions) are the unique solution to the strictly convex optimization problem[2]:

$$\min_{x_{h,\Psi_h}} \sum_{j \in \Psi_h} x_{h,j}(\log x_{h,j} - \log Q_j - 1) \tag{B.1a}$$

$$\text{subject to} \quad A_{i,j} x_{h,j} = x_{h,i}^0 \quad \forall i \in \Psi_h^0, \tag{B.1b}$$

where the constraints impose conservation of mass. A is the stoichiometry matrix with entries $A_{i,j}$ corresponding to the number of strands of type i in complex j, and $x_{h,i}^0$ is the total concentration of strand i introduced to test tube h.

To analyze the equilibrium base-pairing properties of all test tubes $h \in \Omega$, the partition function, Q_j, and equilibrium pair probability matrix, P_j, must be calculated for each complex $j \in \Psi$ using $\Theta(|\phi_j|^3)$ dynamic programs[2,49–56]. The equilibrium concentrations, x_{h,Ψ_h} $\forall h \in \Omega$, are calculated by solving a convex programming problem using an efficient trust region method at a cost that is typically negligible by comparison[2]. The overall time complexity to analyze the test tubes in Ω is then $O(|\Psi||\phi|^3_{\text{max}})$, where $|\phi|_{\text{max}}$ is the size of the largest complex.

Evaluation of the multistate test tube ensemble defect, $\mathcal{M}$, requires calculation of the complex partition functions, Q_Ψ, which are used to calculate the equilibrium concentrations, x_{h,Ψ_h} $\forall h \in \Omega$, as well as the equilibrium pair probability matrices, $P_{\Psi^{\text{on}}}$, which are used to calculate the complex ensemble defects, $n_{\Psi^{\text{on}}}$, and the normalized test tube ensemble defects, $\mathcal{M}_\Omega$. Hence, the time complexity to evaluate the design objective function over the set of target test tubes, Ω, is the same as the time complexity to analyze equilibrium base-pairing in Ω.

B.1.3 Test Tube Ensemble Focusing

To reduce the cost of sequence optimization, the set of complexes, Ψ, is partitioned into two disjoint sets:

$$\Psi = \Psi^{\text{active}} \cup \Psi^{\text{passive}},$$

where Ψ^{active} denotes complexes that will be actively designed and Ψ^{passive} denotes complexes that will inherit sequence information from Ψ^{active}. Only the complexes in Ψ^{active} are directly accounted for in the focused test tube ensembles that are used to evaluate candidate sequences. Initially, we set

$$\Psi^{\text{active}} = \Psi^{\text{on}}, \quad \Psi^{\text{passive}} = \Psi^{\text{off}}, \tag{B.2}$$

where

$$\Psi^{\text{on}} \equiv \cup_{h \in \Omega} \Psi^{\text{on}}_h$$

is the set of complexes that appear as on-targets in at least one test tube, and

$$\Psi^{\text{off}} \equiv \Psi - \Psi^{\text{on}}$$

is the set of complexes that appear as off-targets in at least one test tube and do not appear as on-targets in any test tube. Hence, with test tube ensemble focusing, only complexes that are on-targets in at least one test tube are actively designed at the outset of sequence design.

B.1.4 Hierarchical Ensemble Decomposition

To enable efficient estimation of test tube ensemble properties, the structural ensemble Γ_j of each complex $j \in \Psi^{\text{active}}$ is hierarchically decomposed into a (possibly unbalanced) binary tree of conditional subensembles, yielding a forest of decomposition trees. Each complex $j \in \Psi^{\text{active}}$ contributes a single tree to the decomposition forest whether it is contained in one or more tubes $h \in \Omega$. The structural ensemble of each parent node within the forest is decomposed using one or more exclusive split-points to partition the parent nucleotides to its children. See Reference 1 for details on hierarchical ensemble decomposition. Let Λ denote the set of all nodes in the forest. Let Λ_d denote the set of all nodes at depth d.

Structure-Guided Decomposition of On-Target Complexes

At the outset of sequence design, equilibrium base-pairing probabilities are not yet available to guide ensemble decomposition. Instead, structure-guided hierarchical ensemble decomposition is performed (using a single split-point per parent) for each on-target complex $j \in \Psi^{\text{active}}$, yielding a forest of $|\Psi^{\text{on}}|$ decomposition trees. See Reference 1 for details on structure-guided decomposition.

Stop Condition Stringency

In order to build in a tolerance for a basal level of decomposition defect as subsequences are merged moving up the decomposition forest, the stringency of the stop condition (3.3) is increased by a factor of $f_{\text{stringent}} \in (0, 1)$ at each level moving down the decomposition forest:

$$f_d^{\text{stop}} \equiv f_{\text{stop}}(f_{\text{stringent}})^{d-1} \quad \forall d \in \{1, \ldots, D\}.$$

B.1.5 Efficient Estimation of Test Tube Ensemble Properties

During sequence optimization, the design objective function is estimated based on physical quantities calculated efficiently at any depth $d \in \{1, \ldots, D\}$ in the decomposition forest.

Complex Partition Function Estimate

For each complex $j \in \Psi^{\text{active}}$, the complex partition function estimate, $\tilde{Q}_j$, is calculated from conditional partition functions evaluated efficiently at any depth $d \in \{1, \ldots, D\}$ as described in Reference 1.

Complex Pair Probability Matrix Estimate

For each complex $j \in \Psi^{\text{active}}$, the complex pair probability matrix estimate, $\tilde{P}_j$, is calculated from conditional pair probability matrices evaluated efficiently at any depth $d \in \{1, \ldots, D\}$ as described in Reference 1.

Complex Concentration Estimate using Deflated Mass Constraints

For each tube $h \in \Omega$, the complex concentration estimates, $\tilde{x}_{h,\Psi_h^{\text{active}}}$, are calculated using the complex partition function estimates $\tilde{Q}_{\Psi_h^{\text{active}}}$ previously evaluated at any depth $d \in \{1, \ldots, D\}$ as described in Reference 1. Deflated mass constraints are used to model the effect of the neglected off-target complexes in Ψ_h^{passive}.

Complex Ensemble Defect Estimate

For each complex $j \in \Psi^{\text{on}}$, the complex ensemble defect estimate, $\tilde{n}_j$, is calculated using the complex pair probability matrix estimate $\tilde{P}_j$ previously evaluated at any depth $d \in \{1, \ldots, D\}$ as described in Reference 1. For complex j, the contribution of nucleotide a to the complex ensemble defect estimate is given by:

$$\tilde{n}_j^a = 1 - \sum_{1 \leq b \leq |\phi_j|+1} \tilde{P}_j^{a,b} S_j^{a,b}$$

and the complex ensemble defect estimate is then:

$$\tilde{n}_j = \sum_{1 \leq a \leq |\phi_j|} \tilde{n}_j^a. \tag{B.3}$$

Test Tube Ensemble Defect Estimate

For each tube $h \in \Omega$, the test tube ensemble defect estimate based on $\tilde{x}_{h,\Psi_h^{\text{active}}}$ and $\tilde{n}_{\Psi_h^{\text{on}}}$ calculated at any depth $d \in \{1, \ldots, D\}$, is:

$$\tilde{C}_h = \sum_{j \in \Psi_h^{\text{on}}} \tilde{c}_{h,j}, \tag{B.4}$$

where

$$\tilde{c}_{h,j} = \tilde{n}_j \min\left(\tilde{x}_{h,j}, y_{h,j}\right) + |\phi_j| \max\left(y_{h,j} - \tilde{x}_{h,j}, 0\right) \tag{B.5}$$

is the contribution of complex j. The normalized test tube ensemble defect estimate for tube $h \in \Omega$ at depth $d \in \{1, \ldots, D\}$ is then:

$$\tilde{\mathcal{M}}_h = \tilde{C}_h / y_h^{\text{nt}}, \tag{B.6}$$

where

$$y_h^{\text{nt}} = \sum_{j \in \Psi_h^{\text{on}}} |\phi_j| y_{h,j}$$

is the total concentration of nucleotides in tube h.

Multistate Test Tube Ensemble Defect Estimate

For the set of target test tubes Ω, the objective function estimate based on $\tilde{\mathcal{M}}_\Omega$ evaluated at any depth $d \in \{1, \ldots, D\}$ is then:

$$\tilde{\mathcal{M}} = \frac{1}{|\Omega|} \sum_{h \in \Omega} \tilde{\mathcal{M}}_h. \tag{B.7}$$

We write $\tilde{\mathcal{M}}_d$ in subsequent equations where it is helpful note the depth d at which $\tilde{\mathcal{M}}$ was calculated.

Note that equations (B.3)–(B.7) may be collected into the single equation:

$$\tilde{\mathcal{M}} = \sum_{h \in \Omega} \sum_{j \in \Psi_h^{\text{on}}} \sum_{1 \leq a \leq |\phi_j|} \tilde{\mathcal{M}}_{h,j}^a \tag{B.8}$$

where

$$\tilde{\mathcal{M}}^a_{h,j} \equiv \frac{1}{|\Omega| y^{\rm nt}_h} \left[\tilde{n}^a_j \min(\tilde{x}_{h,j}, y_{h,j}) + \max(y_{h,j} - \tilde{x}_{h,j}, 0) \right], \tag{B.9}$$

is the contribution of nucleotide a in complex $j \in \Psi^{\rm on}_h$ in tube $h \in \Omega$ to the multistate test tube ensemble defect estimate, $\tilde{\mathcal{M}}$, evaluated at any depth $d \in \{1, \ldots, D\}$. This representation is convenient when defining objective function weights (Section B.1.6) and when defining defect-weighted mutation sampling during leaf mutation and defect-weighted reseeding during leaf reoptimization (Sections B.1.7 and B.1.7).

B.1.6 Adjusting Design Priorities using Defect Weights

The user may adjust design priorities by specifying weights for contributions to the multistate test tube ensemble defect estimate, $\tilde{\mathcal{M}}$:

- **Nucleotide weight:** $w^a_{h,j}$ weights the contribution of nucleotide a in complex $j \in \Psi^{\rm on}_h$ in tube $h \in \Omega$.
- **Complex weight:** $w_{h,j}$ weights the contribution of complex $j \in \Psi^{\rm on}_h$ in tube $h \in \Omega$ (equivalent to setting $w^a_{h,j}$ for all nucleotides $1 \le a \le |\phi_j|$).
- **Test tube weight:** w_h weights the contribution of tube $h \in \Omega$ (equivalent to setting $w_{h,j}$ for all complexes $j \in \Psi^{\rm on}_h$).

Each weight takes a value in the interval $[0, \infty)$. By default, all weights are unity. Increasing the weight for a nucleotide, complex, or test tube will lead to a corresponding increase in the allocation of effort to designing this entity, typically leading to a corresponding reduction in the defect contribution of the entity. Likewise, decreasing the weight for a nucleotide, complex, or test tube will lead to a corresponding decrease in the allocation of effort to designing this entity, typically leading to a corresponding increase in the defect contribution of the entity.

Weights are incorporated into the objective function by replacing the defect contribution (B.9) with the weighted defect contribution

$$\tilde{\mathcal{M}}^a_{h,j} \equiv \frac{w_h w_{h,j} w^a_{h,j}}{|\Omega| y^{\rm nt}_h} \left[\tilde{n}^a_j \min(\tilde{x}_{h,j}, y_{h,j}) + \max(y_{h,j} - \tilde{x}_{h,j}, 0) \right], \tag{B.10}$$

and summing using (B.8) as before. If desired, the user can set weights at all three levels, leading to a multiplicative effect. The complex weights and test tube weights exist purely for convenience, as their effects can always be replicated by appropriately setting nucleotide weights (more tediously).

B.1.7 Sequence Optimization at the Leaves of the Decomposition Forest

Initialization

At the outset of sequence optimization, sequences are randomly initialized subject to the constraints in $\mathcal{R}$ by solving a constraint satisfaction problem using a branch and propagate algorithm (Section B.1.11).

Leaf Mutation

To minimize computational cost, all candidate mutation sets are evaluated at the leaf nodes, $k \in \Lambda_D$, of the decomposition forest. Leaf mutation terminates if the *leaf stop condition*,

$$\tilde{\mathcal{M}}_D \leq f_D^{\text{stop}}, \tag{B.11}$$

is satisfied. Here, $\tilde{\mathcal{M}}_D$ denotes the objective function estimated at level D. A candidate mutation set is accepted if it decreases the objective function estimate (B.8) and rejected otherwise.

We perform *defect weighted mutation sampling* by selecting nucleotide a in complex $j \in \Psi_h^{\text{on}}$ in tube $h \in \Omega$ for mutation with probability,

$$\tilde{\mathcal{M}}_{h,j}^{a} / \tilde{\mathcal{M}}_D, \tag{B.12}$$

proportional to its contribution to the objective function. After selecting a candidate mutation position, a candidate mutation is randomly selected from the set of permitted nucleotides at that position. If the resulting sequence is infeasible (due to constraint violations caused by the candidate mutation), a feasible candidate sequence, $\hat{\phi}_{\Lambda_D}$, is generated by solving a constraint satisfaction problem using a branch and propagate algorithm (Section B.1.11).

A feasible candidate sequence, $\hat{\phi}_{\Lambda_D}$, is evaluated via calculation of the objective function estimate, $\tilde{\mathcal{M}}_D$, if the candidate mutation set, ξ, is not in the set of previously rejected mutation sets, γ_{bad}. The set, γ_{bad}, is updated after each unsuccessful mutation and cleared after each successful mutation. The counter m_{bad} is used to keep track of the number of consecutive failed mutation attempts; it is incremented after each unsuccessful mutation and reset to zero after each successful mutation. Leaf mutation terminates unsuccessfully if $m_{\text{bad}} \geq M_{\text{bad}}$. The outcome of leaf mutation is the feasible sequence, ϕ_{Λ_D}, corresponding to the lowest encountered $\tilde{\mathcal{M}}_D$.

Leaf Reoptimization

After leaf mutation terminates, if the leaf stop condition (B.11) is not satisfied, leaf reoptimization commences. At the outset of each round of leaf reoptimization, we perform *defect-weighted reseeding* of M_{reseed} positions by selecting nucleotide a for reseeding (with a new random initial sequence) with probability (B.12). Following reseeding, a feasible candidate sequence, $\hat{\phi}_{\Lambda_D}$, is generated by solving a constraint satisfaction problem using a branch and propagate algorithm (Section B.1.11). After a new round of leaf mutation starting from this reseeded feasible sequence, the reoptimized candidate sequence, $\hat{\phi}_{\Lambda_D}$, is accepted if it decreases $\tilde{\mathcal{M}}_D$ and rejected otherwise. The counter m_{reopt} is used to keep track of the number of rounds of leaf reoptimization; m_{reopt} is incremented after each rejection and reset to zero after each acceptance. Leaf reoptimization terminates successfully if the leaf stop condition is satisfied and unsuccessfully if $m_{\text{reopt}} \geq M_{\text{reopt}}$. The outcome of leaf reoptimization is the feasible sequence, ϕ_{Λ_D}, corresponding to the lowest encountered $\tilde{\mathcal{M}}_D$.

B.1.8 Subsequence Merging, Redecomposition, and Reoptimization

Moving down the decomposition forest, hierarchical ensemble decomposition makes the assumption that base pairs sandwiching parental split-points form with probability approaching unity. Conditional child ensembles enforce these sandwiching base pairs at all levels in the decomposition forest in accordance with the decomposition assumption. As subsequences are merged moving up the decomposition forest, the accuracy of the decomposition assumption is checked. If the assumption is correct, the child-estimated defect will accurately predict the parent-estimated defect. If the assumption is incorrect, the child-estimated defect will not accurately predict the parent-estimated defect since the conditional child ensembles neglect the contributions of structures that lack the sandwiching base pairs. During subsequence merging, if the decomposition assumption is discovered to be incorrect, hierarchical ensemble redecomposition is performed based on the newly available parental base-pairing information. The details of subsequence merging, redecomposition, and reoptimization are as follows.

After leaf reoptimization terminates, parent nodes at depth $d = D - 1$ merge their left and right child sequences to create the candidate sequence $\hat{\phi}_{\Lambda_d}$. The parental objective function estimate, $\tilde{\mathcal{M}}_d$, is calculated and the candidate sequence, $\hat{\phi}_{\Lambda_d}$, is accepted if it decreases $\tilde{\mathcal{M}}_d$ and rejected otherwise. If the *parental stop condition*

$$\tilde{\mathcal{M}}_d \le \max(f_d^{\text{stop}}, \tilde{\mathcal{M}}_{d+1}/f_{\text{stringent}}) \tag{B.13}$$

is satisfied, merging continues up to the next level in the forest. Otherwise, failure to satisfy the parental stop condition indicates the existence of the *decomposition defect*,

$$\tilde{\mathcal{M}}_d - \tilde{\mathcal{M}}_{d+1}/f_{\text{stringent}} > 0,$$

exceeding the basal level permitted by the parameter $f_{\text{stringent}}$. The parent node at depth d whose replacement by its children results in the greatest underestimate of the objective function at level d is subjected to structure- and probability-guided hierarchical ensemble decomposition (Section B.1.10). Additional parents are redecomposed until

$$\tilde{\mathcal{M}}_d - \tilde{\mathcal{M}}^*_{d+1}/f_{\text{stringent}} \le f_{\text{redecomp}}(\tilde{\mathcal{M}}_d - \tilde{\mathcal{M}}_{d+1}/f_{\text{stringent}})$$

where $\tilde{\mathcal{M}}_{d+1}$ is the child defect estimate before any redecomposition, $\tilde{\mathcal{M}}^*_{d+1}$ is the child defect estimate after redecomposition, and $f_{\text{redecomp}} \in (0, 1)$.

After redecomposition, the current sequences at depth d are pushed to level D, the lowest encountered defect estimate is reset for all levels below d, and a new round of leaf mutation and leaf reoptimization is performed. Following leaf reoptimization, merging begins again. Subsequence merging and reoptimization terminate successfully if the parental stop condition (B.13) is satisfied at depth $d = 1$. The outcome of subsequence merging, redecomposition, and reoptimization is the feasible sequence, ϕ_{Λ_1}, corresponding to the lowest encountered $\tilde{\mathcal{M}}_1$.

B.1.9 Test Tube Evaluation, Refocusing, and Reoptimization

Using test tube ensemble focusing, initial sequence optimization is performed for the on-target complexes in Ψ^{active}, neglecting the off-target complexes in Ψ^{passive}. At the termination of initial forest optimization, the estimated design objective function is $\tilde{\mathcal{M}}_1$, calculated using (B.8). The estimated contributions for each tube $h \in \Omega$ are based on complex concentration estimates, $\tilde{x}_{h,\Psi_h^{\text{active}}}$, calculated using deflated total strand concentrations (equation (10) of Reference 1) to create a built-in defect allowance for the effect of the neglected off-targets in Ψ_h^{passive}. The exact design objective function, $\mathcal{M}$, is then evaluated for the first time over the full ensemble Ψ. For this exact calculation, the objective function, $\mathcal{M}$, is based on complex concentrations, x_{h,Ψ_h}, calculated using the full strand concentrations (equation (9) of Reference 1).

If the objective function satisfies the *termination stop condition*,

$$\mathcal{M} \leq \max(f_{\text{stop}}, \tilde{\mathcal{M}}_1), \tag{B.14}$$

sequence design terminates successfully. Otherwise, failure to satisfy the termination stop condition indicates the existence of the *focusing defect*,

$$\mathcal{M} - \tilde{\mathcal{M}}_1 > 0. \tag{B.15}$$

The multistate test tube ensemble is refocused by transferring the highest-concentration off-target in Ψ^{passive} to Ψ^{active}. Additional off-targets are transferred from Ψ^{passive} to Ψ^{active} until

$$\mathcal{M} - \tilde{\mathcal{M}}_1^* \leq f_{\text{refocus}}(\mathcal{M} - \tilde{\mathcal{M}}_1), \tag{B.16}$$

where $\tilde{\mathcal{M}}_1$ is the forest-estimated defect before any refocusing, $\tilde{\mathcal{M}}_1^*$ is the forest-estimated defect after refocusing (calculated using deflated total strand concentrations (equation (10) of Reference 1) if $\Psi^{\text{passive}} \neq \emptyset$), and $f_{\text{refocus}} \in (0, 1)$.

The new off-targets in Ψ^{active} are then decomposed using probability-guided hierarchical ensemble decomposition (Section B.1.10), the decomposition forest is augmented with new nodes at all depths, and forest reoptimization commences starting from the final sequences from the previous round of forest optimization. During forest reoptimization, the algorithm actively attempts to destabilize the off-targets that were added to Ψ^{active}. This process of test tube ensemble refocusing and forest reoptimization is repeated until the termination stop condition (B.14) is satisfied, which is guaranteed to occur in the event that all off-targets are eventually added to Ψ^{active}. At the conclusion of sequence design, the algorithm returns the feasible sequence set, ϕ_Ψ, that yielded the lowest encountered objective function, $\mathcal{M}$.

B.1.10 Hierarchical Ensemble Decomposition Using Multiple Exclusive Split-Points

Prior to sequence optimization, in the absence of base-pairing probability information, hierarchical ensemble decomposition is performed for each complex $j \in \Psi^{\text{active}}$ based on user-specified target structures. During subsequence merging, if decomposition defects are encountered, or during test tube evaluation, if focusing defects are encountered, subsequent hierarchical ensemble decomposition takes advantage of the newly available parental base-pairing probabilities. In either case, selection of the optimal set of exclusive split-points is determined using a branch and bound algorithm to minimize the cost of evaluating the child nodes (see Section S1.4 of Reference 1).

Probability-Guided Decomposition using Multiple Exclusive Split-Points

During redecomposition (Section B.1.8) and refocusing (Section B.1.9), parent nodes that lack a target structure are decomposed via probability-guided decomposition using multiple exclusive split-points. See Reference 1 for details on probability-guided decomposition.

Structure- and Probability-Guided Decomposition using Multiple Exclusive Split-Points

During redecomposition (Section B.1.8), parent nodes that have a target structure are decomposed via structure- and probability-guided decomposition using multiple exclusive split-points. See Reference 1 for details on structure- and probability-guided decomposition.

Multistate Test Tube Ensemble Defect Estimate Using Multiple Exclusive Decompositions

Because exclusive split-points lead to exclusive structural ensembles, the expressions used to estimate ensemble properties over Ω (Section B.1.5) can be generalized to account for the possibility of multiple exclusive split-points within any parent in the decomposition forest. See Reference 1 for details.

B.1.11 Generation of Feasible Sequences

Each time the sequence is initialized, mutated, or reseeded, a feasible sequence is generated by solving a constraint satisfaction problem based on the user-specified constraints in $\mathcal{R}$.

Constraint Satisfaction Problem

A constraint satisfaction problem (CSP)[57] is specified as:

- a set of *variables*,
- a set of *domains*, each listing the possible values for the corresponding variable,
- a set of *constraints*, each defined by a constraint relation operating on a subset of the variables.

In the present setting, each variable is the sequence, ϕ^a, of a nucleotide, a. For RNA, the domain for each variable is $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$. Each constraint in $\mathcal{R}$ is specified using one of the constraint relations in Table 3.1 applied to one or more nucleotides (e.g., specification of constraint $R^{\text{match}}_{a,b}$ requires that $\phi^a = \phi^b$ for nucleotides a and b).

Table B.1: IUPAC degenerate nucleotide codes for RNA.

Code	Nucleotides
M	A or C
R	A or G
W	A or U
S	C or G
Y	C or U
K	G or U
V	A, C, or G
H	A, C, or U
D	A, G, or U
B	C, G, or U
N	A, C, G, or U

T replaces U for DNA.

In general, constraint satisfaction problems are NP-complete, so general-purpose polynomial-time algorithms are unavailable[57]. Empirically, we find that CSPs arising in the context of nucleic acid reaction pathway engineering specified in terms of the diverse constraint relations of Table 3.1 can typically be solved efficiently using the branch and propagate algorithm described below.

Branch and Propagate Algorithm

We solve the CSP using a branch and propagate algorithm that returns a solution if one exists and returns a warning if no solution exists. Initially, the domain for each variable is $\{A, C, G, U\}$. We first pre-process the CSP by trivially removing any value from the domain of a variable a that is inconsistent with a constraint (e.g., an assignment or library constraint). We further pre-process the CSP using constraint propagation to impose arc consistency as described below.

The branch and propagate algorithm involves iterated application of two ingredients:

- *constraint propagation* is used to narrow the search space by imposing *arc consistency* on each pair of variables: for any value in the domain of variable a there must be a consistent value in the domain of every other variable b, otherwise that value of variable a is inconsistent and can be removed from the domain of a (see "Chapter 3: Consistency-Enforcing and Constraint Propagation" of Reference 57).
- *depth-first branching* is used to extend a candidate partial solution by assigning a consistent value to one additional variable a, followed by *backtracking* to reassign the value of the most-recently assigned variable if no value in the domain of a is consistent with previous assignments (see "Chapter 5: General Search Strategies: Look-Ahead" of Reference 57).

Feasible Sequence Inititialization

Sequence initialization (Section B.1.7) commences with constraint propagation to impose arc consistency and then a first branching step in which a variable, a, is randomly selected and randomly assigned a value from the domain of a. Constraint propagation is then used to impose arc consistency and the next branching step is taken by randomly selecting an unassigned variable, b, and assigning a consistent value from the domain of b. Backtracking is performed if no consistent value for b exists. The branch and propagate algorithm returns a feasible set of initial sequences, $\phi_{\Psi^{\text{active}}}$, if one exists, and a warning otherwise.

Feasible Sequence Mutation

During leaf mutation (Section B.1.7), a feasible candidate sequence is generated by mutating the current leaf sequence, ϕ_{Λ_D}. This process begins with the sequence design algorithm randomly selecting a nucleotide a for mutation with probability (B.12) and randomly assigning a new value from the domain of a. We then solve a CSP to obtain a valid candidate sequence consistent with the new value of a. Constraint propagation is used to impose arc consistency with the new value of a and any variables that require reassignment are added to the candidate mutation set, ξ_a. Initially, branching is performed by randomly selecting an unassigned variable b from ξ_a with probability proportional to the size of the domain of b (i.e., using weight

$$w_b = |\text{domain}(b)| \tag{B.17}$$

to calculate the probability of selecting b). For each value ϕ^b in the domain of b, we check the implications of arc consistency on the size of the candidate mutation set, $\xi_{a,b}$, and create a priority queue based on the minimum increase in $|\xi_{a,b}|$ relative to $|\xi_a|$. Let $|\xi_{a,b_1}|$ denote the minimum increase, $|\xi_{a,b_2}|$ denote the next largest increase, and so on. Branching is performed by exploring the values of b according to their rank order in this priority queue. If no consistent value of b exists, backtracking is performed and the selection weight for variable b (B.17) is updated using

$$w_b = \epsilon w_b + (1-\epsilon)(|\xi_{a,b_1}| - |\xi_a|). \tag{B.18}$$

where $\epsilon = 0.5$ is a decay constant. The heuristics (B.17) and (B.18) seek to preferentially select highly constrained variables early in the branching process to avoid excessive backtracking. The initial weights (B.17) assume that each variable b will imply a mutation set $\xi_{a,b}$ that increases in size with the size of domain(b), preferentially selecting variables with larger domains. With (B.18), as we explore different variables b in ξ_a, we explicitly calculate the implied increase in $\xi_{a,b}$ due to each branching decision, and update the weights to bias future searching toward selection of b variables that cause the highest minimal increase in the size of the mutation set ξ_a.

The branch and propagate algorithm returns a feasible candidate sequence, $\hat{\phi}_{\Lambda_D}$, if one exists. Otherwise, the new value of a is invalid and is removed from the domain of a; a new value of a is randomly selected from the domain of a, and branch and propagate is applied again. If the only valid value of a is the current value, then m_{bad} is incremented and the leaf mutation procedure selects a new nucleotide for mutation with probability (B.12).

Feasible Sequence Reseeding

During leaf reoptimization (Section B.1.7), a feasible candidate reseeded sequence, $\hat{\phi}_{\Lambda_D}$, is generated by introducing M_{reseed} feasible sequence mutations to the current leaf sequence, ϕ_{Λ_D}, via M_{reseed} consecutive calls to the branch and propagate algorithm of Section B.1.11 (selecting nucleotide a for mutation without replacement with probability (B.12)).

B.1.12 Pseudocode

```
$\textsc{OptimizeTubes}(\Omega, \Psi^{\text{on}}_{\Omega}, \Psi^{\text{off}}_{\Omega}, \Psi, s_{\Psi}, y_{\Omega,\Psi}, \mathcal{R})$
    $\Psi^{\text{active}}, \Psi^{\text{passive}} \leftarrow \Psi^{\text{on}}, \Psi^{\text{off}}$
    $\phi_{\Psi^{\text{active}}} \leftarrow \textsc{InitSeq}(s_{\Psi^{\text{active}}}, \mathcal{R})$
    $\Lambda, D \leftarrow \textsc{MakeForest}(s_{\Psi^{\text{active}}})$
    $\phi_{\Lambda}, \tilde{\mathcal{M}}_1 \leftarrow \textsc{OptimizeForest}(\phi_{\Lambda}, D)$
    $\mathcal{M} \leftarrow \textsc{EvaluateDefect}(\phi_{\Psi})$
    $\hat{\phi}_{\Psi}, \hat{\mathcal{M}} \leftarrow \phi_{\Psi}, \mathcal{M}$
    while $\hat{\mathcal{M}} > \max(f_{\text{stop}}, \tilde{\mathcal{M}}_1)$
        $\Psi^{\text{active}}, \Psi^{\text{passive}} \leftarrow \textsc{RefocusTubes}($
                $\Psi^{\text{active}}, \Psi^{\text{passive}}, \{\hat{x}_{h,\Psi_h^{\text{passive}}}\})$
        $\Lambda, D \leftarrow \textsc{AugmentForest}(\Lambda, D, \hat{P}_{\Psi^{\text{active}}})$
        $\hat{\phi}_{\Lambda}, \tilde{\mathcal{M}}_1 \leftarrow \textsc{OptimizeForest}(\hat{\phi}_{\Lambda}, D)$
        $\hat{\mathcal{M}} \leftarrow \textsc{EvaluateDefect}(\hat{\phi}_{\Psi})$
        if $\hat{\mathcal{M}} < \mathcal{M}$
            $\phi_{\Psi}, \mathcal{M} \leftarrow \hat{\phi}_{\Psi}, \hat{\mathcal{M}}$
    return $\phi_{\Psi}$
```

```
$\textsc{OptimizeForest}(\phi_{\Lambda}, D)$
    $\tilde{\mathcal{M}}_d \leftarrow \infty \ \forall d \in \{1, \ldots, D\}$
    $\beta_{\text{merge}} \leftarrow$ false
    while $\neg\beta_{\text{merge}}$
        $\phi_{\Lambda_D}, \tilde{\mathcal{M}}_D \leftarrow \textsc{OptimizeLeaves}(\phi_{\Lambda_D}, D)$
        $d \leftarrow D - 1$
        $\beta_{\text{merge}} \leftarrow$ true
        while $d \geq 1$ and $\beta_{\text{merge}}$
            $\hat{\phi}_{\Lambda_d} \leftarrow \textsc{MergeSeq}(\phi_{\Lambda_{d+1}})$
            $\hat{\mathcal{M}}_d \leftarrow \textsc{EstimateDefect}(\hat{\phi}_{\Lambda_d})$
            if $\hat{\mathcal{M}}_d < \tilde{\mathcal{M}}_d$
                $\phi_{\Lambda_d}, \tilde{\mathcal{M}}_d \leftarrow \hat{\phi}_{\Lambda_d}, \hat{\mathcal{M}}_d$
            if $\hat{\mathcal{M}}_d > \max(f_d^{\text{stop}}, \tilde{\mathcal{M}}_{d+1}/f_{\text{stringent}})$
                $\beta_{\text{merge}} \leftarrow$ false
                $\Lambda, D \leftarrow \textsc{RedecomposeForest}($
                        $\Lambda, D, s_{\Lambda_d}, \hat{P}_{\Lambda_d})$
                $\phi_{\Lambda_D} \leftarrow \textsc{SplitSeq}(\hat{\phi}_{\Lambda_d})$
                $\tilde{\mathcal{M}}_{d'} \leftarrow \infty \ \forall d' \in \{d+1, \ldots, D\}$
            $d \leftarrow d - 1$
    return $\phi_{\Lambda_1}, \tilde{\mathcal{M}}_1$
```

```
$\textsc{OptimizeLeaves}(\phi_{\Lambda_D}, D)$
    $\phi_{\Lambda_D}, \tilde{\mathcal{M}}_D \leftarrow \textsc{MutateLeaves}(\phi_{\Lambda_D}, D)$
    $m_{\text{reopt}} \leftarrow 0$
    while $\tilde{\mathcal{M}}_D > f_D^{\text{stop}}$ and $m_{\text{reopt}} < M_{\text{reopt}}$
        $\hat{\phi}_{\Lambda_D} \leftarrow \textsc{ReseedSeq}(\phi_{\Lambda_D}, \{\tilde{\mathcal{M}}^a_{h,j}\}, \mathcal{R})$
        $\hat{\phi}_{\Lambda_D}, \hat{\mathcal{M}}_D \leftarrow \textsc{MutateLeaves}(\hat{\phi}_{\Lambda_D}, D)$
        if $\hat{\mathcal{M}}_D < \tilde{\mathcal{M}}_D$
            $\phi_{\Lambda_D}, \tilde{\mathcal{M}}_D \leftarrow \hat{\phi}_{\Lambda_D}, \hat{\mathcal{M}}_D$
            $m_{\text{reopt}} \leftarrow 0$
        else
            $m_{\text{reopt}} \leftarrow m_{\text{reopt}} + 1$
    return $\phi_{\Lambda_D}, \tilde{\mathcal{M}}_D$
```

```
$\textsc{MutateLeaves}(\phi_{\Lambda_D}, D)$
    $\tilde{\mathcal{M}}_D \leftarrow \textsc{EstimateDefect}(\phi_{\Lambda_D})$
    $\gamma_{\text{bad}} \leftarrow \emptyset, \quad m_{\text{bad}} \leftarrow 0$
    while $\tilde{\mathcal{M}}_D > f_D^{\text{stop}}$ and $m_{\text{bad}} < M_{\text{bad}}$
        $\xi, \hat{\phi}_{\Lambda_D} \leftarrow \textsc{SampleMutation}(\phi_{\Lambda_D}, \{\tilde{\mathcal{M}}^a_{h,j}\}, \mathcal{R})$
        if $\xi \in \gamma_{\text{bad}}$
            $m_{\text{bad}} \leftarrow m_{\text{bad}} + 1$
        else
            $\hat{\mathcal{M}}_D \leftarrow \textsc{EstimateDefect}(\hat{\phi}_{\Lambda_D})$
            if $\hat{\mathcal{M}}_D < \tilde{\mathcal{M}}_D$
                $\phi_{\Lambda_D}, \tilde{\mathcal{M}}_D \leftarrow \hat{\phi}_{\Lambda_D}, \hat{\mathcal{M}}_D$
                $\gamma_{\text{bad}} \leftarrow \emptyset, \quad m_{\text{bad}} \leftarrow 0$
            else
                $\gamma_{\text{bad}} \leftarrow \gamma_{\text{bad}} \cup \xi, \quad m_{\text{bad}} \leftarrow m_{\text{bad}} + 1$
    return $\phi_{\Lambda_D}, \tilde{\mathcal{M}}_D$
```

```
$\textsc{EstimateDefect}(\phi_{\Lambda_d})$
    $\tilde{Q}_{\Lambda_d}, \tilde{P}_{\Lambda_d} \leftarrow \textsc{ConditionalNodalProperties}(\phi_{\Lambda_d})$
    $\tilde{Q}_{\Psi^{\text{active}}} \leftarrow \textsc{EstimateComplexPfuncs}(\tilde{Q}_{\Lambda_d})$
    $\tilde{P}_{\Psi^{\text{active}}} \leftarrow \textsc{EstimateComplexPairProbs}(\tilde{P}_{\Lambda_d})$
    for $h \in \Omega$
        $\tilde{x}^0_{h,\Psi^0_h} \leftarrow \textsc{DeflateMassConstraints}(x^0_{h,\Psi^0_h})$
        $\tilde{x}_{h,\Psi^{\text{active}}_h} \leftarrow \textsc{EstimateComplexConcs}(\tilde{Q}_{\Psi^{\text{active}}_h}, \tilde{x}^0_{h,\Psi^0_h})$
        $\{\tilde{\mathcal{M}}^a_{h,j}\} \leftarrow \textsc{EstimateContribs}(\tilde{P}_{\Psi^{\text{on}}_h}, s_{\Psi^{\text{on}}_h}, \tilde{x}_{h,\Psi^{\text{on}}_h}, y_{h,\Psi^{\text{on}}_h})$
    $\tilde{\mathcal{M}}_d \leftarrow \sum_{h\in\Omega} \sum_{j\in\Psi^{\text{on}}_h} \sum_{1\leq a\leq|\phi_j|} \tilde{\mathcal{M}}^a_{h,j}$
    return $\tilde{\mathcal{M}}_d$
```

Algorithm B.1: Pseudocode for constrained multistate test tube ensemble defect optimization. Consider the set of target test tubes, Ω, collectively containing the set of complexes Ψ (comprising the sets of on-target complexes $\Psi^{\text{on}}_{\Omega}$ and off-target complexes $\Psi^{\text{off}}_{\Omega}$) with target secondary structures s_{Ψ} and target concentrations $y_{\Omega,\Psi}$. The function call $\textsc{OptimizeTubes}(\Omega, \Psi^{\text{on}}_{\Omega}, \Psi^{\text{off}}_{\Omega}, \Psi, s_{\Psi}, y_{\Omega,\Psi}, \mathcal{R})$ returns the set of designed sequences, ϕ_{Ψ}, satisfying the sequence constraints in $\mathcal{R}$.

B.1.13 Default Algorithm Parameters

Default algorithm parameters are shown in Table B.2.

Table B.2: RNA design: default algorithm parameters for constrained multistate test tube ensemble defect optimization.

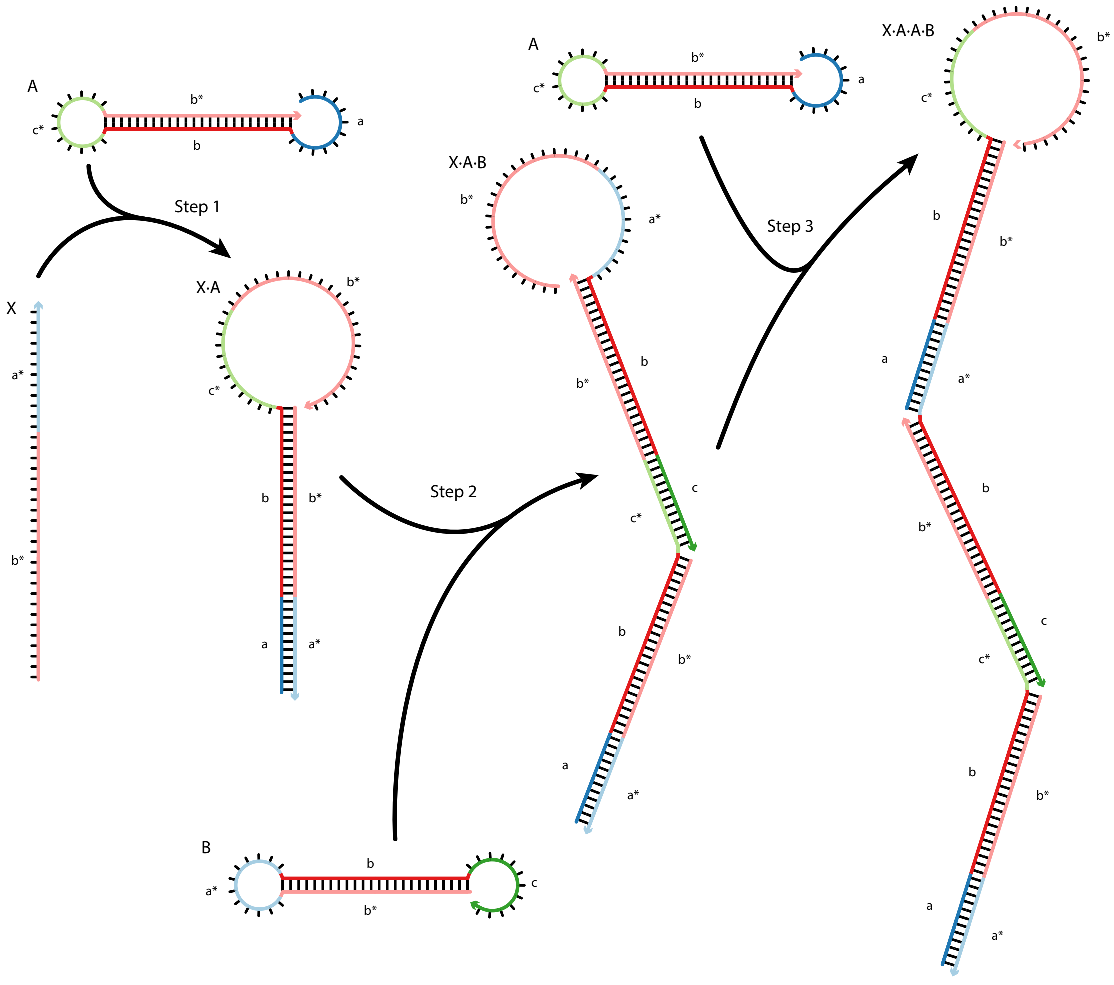
Parameter	Value
f_{stop}	0.02
f_{passive}	0.01
H_{split}	2
N_{split}	12
f_{split}	0.99
$f_{\text{stringent}}$	0.99
ΔG^{clamp}	-25 kcal/mol
M_{bad}	300
M_{reseed}	50
M_{reopt}	3
f_{redecomp}	0.03
f_{refocus}	0.03

For DNA design, $H_{\text{split}} = 3$.

B.2 Engineering Case Studies

B.2.1 Reaction Pathways

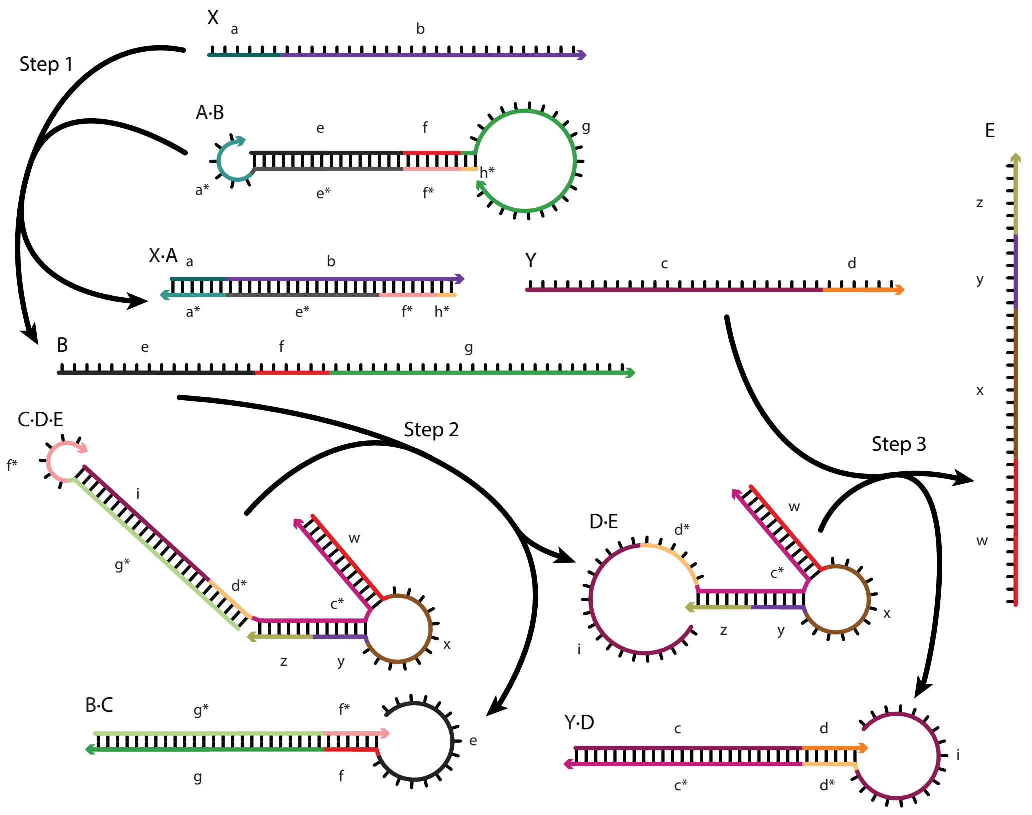
Conditional Self-Assembly via Hybridization Chain Reaction (HCR)



Step	Reaction	Function	Mechanism
1	X + A → X·A	detect target, first A polymerization step	toehold/toehold nucleation, 3-way branch migration
2	X·A + B → X·A·B	first B polymerization step, regenerate target sequence	toehold/toehold nucleation, 3-way branch migration
2k+1	$\mathrm{X}\cdot(\mathrm{A})_k\cdot(\mathrm{B})_k + \mathrm{A} \rightarrow \mathrm{X}\cdot(\mathrm{A})_{k+1}\cdot(\mathrm{B})_k$	generic A polymerization step, k = 1, 2, ...	toehold/toehold nucleation, 3-way branch migration
2k+2	$\mathrm{X}\cdot(\mathrm{A})_{k+1}\cdot(\mathrm{B})_k + \mathrm{B} \rightarrow \mathrm{X}\cdot(\mathrm{A})_{k+1}\cdot(\mathrm{B})_{k+1}$	generic B polymerization step, k = 1, 2, ...	toehold/toehold nucleation, 3-way branch migration

Figure B.1: Reaction pathway for conditional self-assembly via hybridization chain reaction (HCR)[19]. Target X triggers self-assembly of metastable hairpins A and B into a long nicked dsDNA polymer via a chain reaction of alternating A and B polymerization steps. Top: Reaction pathway schematic. Bottom: Elementary step details.

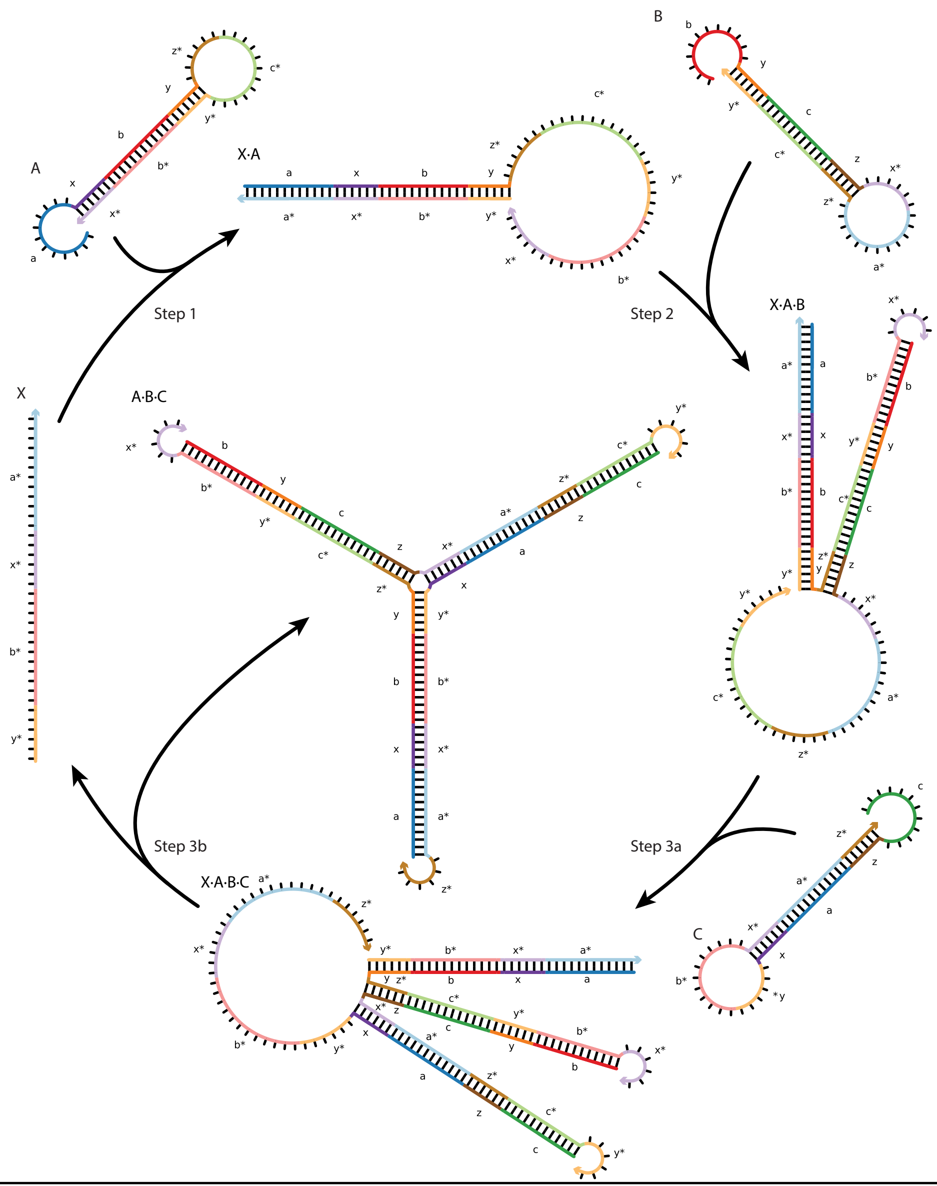
Boolean Logic AND using Toehold Sequestration Gates



Step	Reaction	Function	Mechanism
1	X + A·B → X·A + B	translate X target sequence	toehold/toehold nucleation, 3-way branch migration
2	B + C·D·E → B·C + D·E	detect translated first target	toehold/toehold nucleation, 3-way branch migration, expose toehold
3	Y + D·E → Y·D + E	detect second target	toehold/toehold nucleation, 3-way branch migration

Figure B.2: Reaction pathway for Boolean logic AND using toehold sequestration gates[58]. Gates implement the logical operation "if targets X AND Y are detected, generate output E". Top: Reaction pathway schematic. Bottom: Elementary step details.

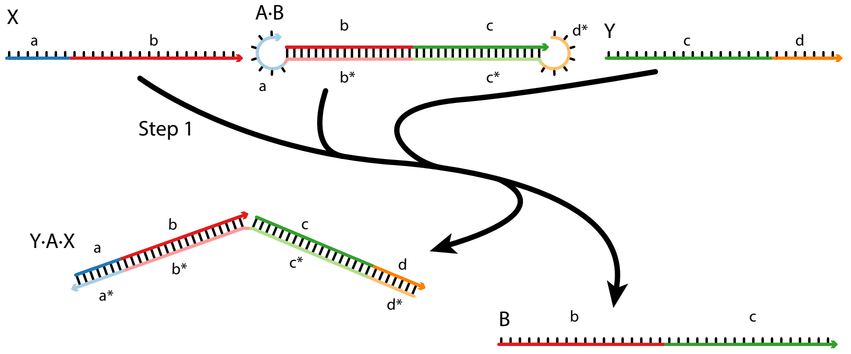
Self-Assembly of a 3-Arm Junction via Catalytic Hairpin Assembly (CHA)



Step	Reaction	Function	Mechanism
1	X + A → X·A	assemble with catalyst X	toehold/toehold nucleation, 3-way branch migration
2	X·A + B → X·A·B	assemble	toehold/toehold nucleation, 3-way branch migration
3a	X·A·B + C → X·A·B·C	assemble	toehold/toehold nucleation, 3-way branch migration
3b	X·A·B·C → X + A·B·C	disassemble from catalyst X and assemble 3-arm junction	intracomplex blunt-end strand invasion, 3-way branch migration

Figure B.3: Reaction pathway for self-assembly of a 3-arm junction via catalytic hairpin assembly (CHA)[23]. Target X catalyzes self-assembly of metastable hairpins A, B, and C into 3-arm junction A·B·C. Top: Reaction pathway schematic. Bottom: Elementary step details.

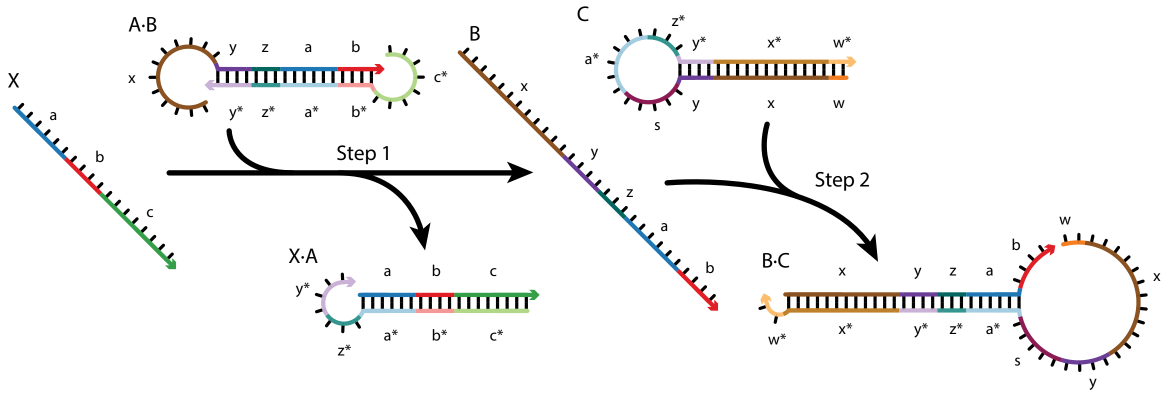
Boolean Logic AND using a Cooperative Hybridization Gate



Step	Reaction	Function	Mechanism
1	X + Y + A·B → Y·A·X + B	cooperative detection of X and Y to generate output B	toehold/toehold nucleation, 3-way branch migration

Figure B.4: Reaction pathway for Boolean logic AND using a cooperative hybridization gate[59]. Gate implements the logical operation "if targets X AND Y are detected, cooperatively generate output B". Top: Reaction pathway schematic. Bottom: Elementary step details.

Conditional Dicer Substrate Formation via Shape and Sequence Transduction with Small Conditional RNAs (scRNAs)



Step	Reaction	Function	Mechanism
1	X + A·B → X·A + B	detect target X (sequence ‘a-b-c’)	toehold/toehold nucleation, 3-way branch migration, spontaneous dissociation
2	B + C → B·C	form Dicer substrate targeting independent target Y (sequence ‘w-x-y-z’)	toehold/loop nucleation, 3-way branch migration

Figure B.5: Reaction pathway for conditional Dicer substrate formation via shape and sequence transduction with small conditional RNAs (scRNAs)[32]. scRNA A·B detects target X (comprising sequence ‘a-b-c’), generating intermediate B that assembles with scRNA C to generate Dicer substrate B·C (targeting independent sequence ‘w-x-y-z’ for silencing). Top: Reaction pathway schematic. Bottom: Elementary step details.

Conditional Self-Assembly via HCR

Target test tubes are defined using the specification of Section 3.2.3 with the following definitions. The total number of target test tubes is $|\Omega| = \sum_{n=1,\ldots,N}\{\text{Step 0, Step 1, Step 2, Step 3}\}_n + \text{Crosstalk} = 4N+1$; the target test tubes in the multistate test tube ensemble, Ω, are indexed by $h = 1,\ldots,4N+1$. $L_{\max} = 2$ for all tubes.

Reactants for system *n*

- Target: X_n
- Hairpins: $\{A, B\}_n$

Elementary step tubes for system *n*

- Step 0_n tube: $\Psi^{\text{products}}_{0_n} \equiv \{X, A, B\}_n$; $\Psi^{\text{reactants}}_{0_n} \equiv \{A{\cdot}B\}_n$ (dimer nucleus that inhibits leakage); $\Psi^{\text{exclude}}_{0_n} \equiv \{X{\cdot}A\}_n$ (downstream on-pathway product)
- Step 1_n tube: $\Psi^{\text{products}}_{1_n} \equiv \{X{\cdot}A\}_n$; $\Psi^{\text{reactants}}_{1_n} \equiv \{X, A\}_n$; $\Psi^{\text{exclude}}_{1_n} \equiv \emptyset$
- Step 2_n tube: $\Psi^{\text{products}}_{2_n} \equiv \{X{\cdot}A{\cdot}B\}_n$; $\Psi^{\text{reactants}}_{2_n} \equiv \{X{\cdot}A, B\}_n$; $\Psi^{\text{exclude}}_{2_n} \equiv \emptyset$
- Step 3_n tube: $\Psi^{\text{products}}_{3_n} \equiv \{X{\cdot}A{\cdot}A{\cdot}B\}_n$; $\Psi^{\text{reactants}}_{3_n} \equiv \{X{\cdot}A{\cdot}B, A\}_n$; $\Psi^{\text{exclude}}_{3_n} \equiv \emptyset$

Global crosstalk tube

- Crosstalk tube: $\Psi^{\text{reactive}}_{\text{global}} \equiv \cup_{n=1,\ldots,N}\{\lambda^{\text{reactive}}_n\}$; $\Psi^{\text{crosstalk}}_{\text{global}} \equiv \Psi^{L\le L_{\max}}_{\text{global}} - \cup_{n=1,\ldots,N}\{\lambda^{\text{cognate}}_n\}$

The reactive species and cognate products for system *n* are:

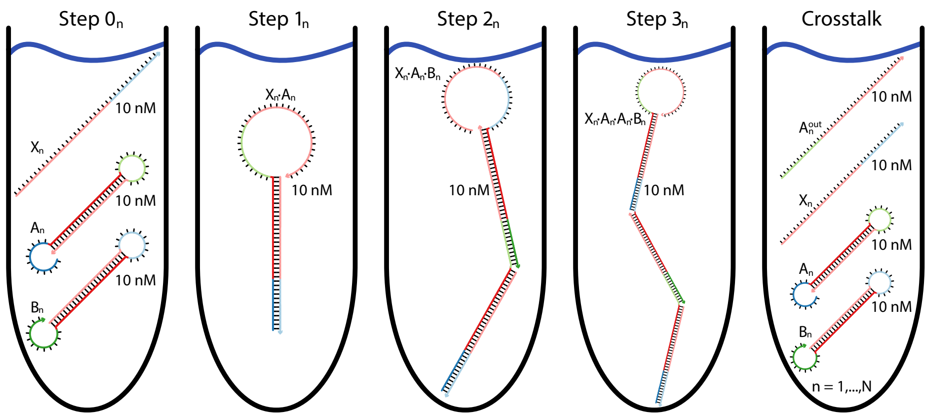
- $\lambda^{\text{simple}}_n \equiv \{A, B\}_n$
- $\lambda^{\text{ss-out}}_n \equiv \{X, A^{\text{out}}, B^{\text{out}}\}_n$
- $\lambda^{\text{ss-in}}_n \equiv \{A^{\text{toe}}, B^{\text{toe}}\}_n$
- $\lambda^{\text{reactive}}_n \equiv \{A, B, A^{\text{out}}, B^{\text{out}}\}_n$
- $\lambda^{\text{cognate}}_n \equiv \{A^{\text{out}}{\cdot}B, B^{\text{out}}{\cdot}A\}_n$

based on the definitions (listed $5'$ to $3'$ using the sequence domain notation of Figure B.1):

- $A \equiv A^{\text{in}}\text{-}A^{\text{out}}$
- $A^{\text{toe}} \equiv a$

- $A^{\text{in}} \equiv$ a-b
- $A^{\text{out}} \equiv$ c*-b*
- $B \equiv B^{\text{out}}$-B^{in}
- $B^{\text{toe}} \equiv$ c
- $B^{\text{in}} \equiv$ b-c
- $B^{\text{out}} \equiv$ b*-a*
- $X \equiv$ b*-a*

Note: X_n is identical to B_n^{out}, so it is implicitly included in the definition of $\lambda_n^{\text{reactive}}$. To avoid redundancy, the toeholds of $\lambda_n^{\text{ss-in}}$ are not included in the definition of $\lambda_n^{\text{reactive}}$; these toeholds are already available to form dimer crosstalk products in the hairpin monomers of $\lambda_n^{\text{simple}}$.



Tube	On-targets (Ψ_h^{on})	Off-targets (Ψ_h^{off})
Step 0_n	$\{\text{X, A, B}\}_n$	$\{\text{A·B}\}_n \cup \Psi_{0_n}^{L\le L_{\max}} - \{\text{X·A}\}_n$
Step 1_n	$\{\text{X·A}\}_n$	$\{\text{X, A}\}_n \cup \Psi_{1_n}^{L\le L_{\max}}$
Step 2_n	$\{\text{X·A·B}\}_n$	$\{\text{X·A, B}\}_n \cup \Psi_{2_n}^{L\le L_{\max}}$
Step 3_n	$\{\text{X·A·A·B}\}_n$	$\{\text{X·A·B, A}\}_n \cup \Psi_{3_n}^{L\le L_{\max}}$
Crosstalk	$\cup_{n=1,\ldots,N}\{\lambda_n^{\text{reactive}}\}$	$\Psi_{\text{global}}^{L\le L_{\max}} - \cup_{n=1,\ldots,N}\{\lambda_n^{\text{cognate}}\}$

Figure B.6: Target test tubes for conditional self-assembly via HCR (reaction pathway of Figure B.1). Top: Target test tube schematics. Bottom: Target test tube details. Each target test tube contains the depicted on-target complexes (each with the depicted target structure and a target concentration of 10 nM) and the off-target complexes listed in the table (each with vanishing target concentration). To simultaneously design N orthogonal systems, the total number of target test tubes is $|\Omega| = 4N + 1$. $L_{\max} = 2$ for all tubes. Design conditions: DNA in 1 M Na$^+$ at 25 °C.

Boolean Logic AND using Toehold Sequestration Gates

Target test tubes are defined using the specification of Section 3.2.3 with the following definitions. The total number of target test tubes is $|\Omega| = \sum_{n=1,\ldots,N} \{\text{Step 0, Step 1, Step 2, Step 3}\}_n + \text{Crosstalk} = 4N+1$; the target test tubes in the multistate test tube ensemble, Ω, are indexed by $h = 1,\ldots,4N+1$. $L_{\max} = 2$ for all tubes.

Reactants for system *n*

- Targets: $\{\text{X, Y}\}_n$
- Translator gate: $\{\text{A}\cdot\text{B}\}_n$
- AND gate: $\{\text{C}\cdot\text{D}\cdot\text{E}\}_n$

Elementary step tubes for system *n*

- Step 0_n: $\Psi^{\text{products}}_{0_n} \equiv \{\text{X, Y, A}\cdot\text{B, C}\cdot\text{D}\cdot\text{E}\}_n$; $\Psi^{\text{reactants}}_{0_n} \equiv \{\text{A, B, C, D, E, C}\cdot\text{D, D}\cdot\text{E}\}_n$; $\Psi^{\text{exclude}}_{0_n} \equiv \{\text{X}\cdot\text{A}\}_n$
- Step 1_n: $\Psi^{\text{products}}_{1_n} \equiv \{\text{X}\cdot\text{A, B}\}_n$; $\Psi^{\text{reactants}}_{1_n} \equiv \{\text{X, A}\cdot\text{B}\}_n$; $\Psi^{\text{exclude}}_{1_n} \equiv \emptyset$
- Step 2_n: $\Psi^{\text{products}}_{2_n} \equiv \{\text{B}\cdot\text{C, D}\cdot\text{E}\}_n$; $\Psi^{\text{reactants}}_{2_n} \equiv \{\text{B, C}\cdot\text{D}\cdot\text{E}\}_n$; $\Psi^{\text{exclude}}_{2_n} \equiv \emptyset$
- Step 3_n: $\Psi^{\text{products}}_{3_n} \equiv \{\text{Y}\cdot\text{D, E}\}_n$; $\Psi^{\text{reactants}}_{3_n} \equiv \{\text{Y, D}\cdot\text{E}\}_n$; $\Psi^{\text{exclude}}_{3_n} \equiv \emptyset$

Global crosstalk tube

- Crosstalk tube: $\Psi^{\text{reactive}}_{\text{global}} \equiv \cup_{n=1,\ldots,N}\{\lambda^{\text{reactive}}_n\}$; $\Psi^{\text{crosstalk}}_{\text{global}} \equiv \Psi^{L\le L_{\max}}_{\text{global}} - \cup_{n=1,\ldots,N}\{\lambda^{\text{cognate}}_n\}$

The reactive species and cognate products for system *n* are:

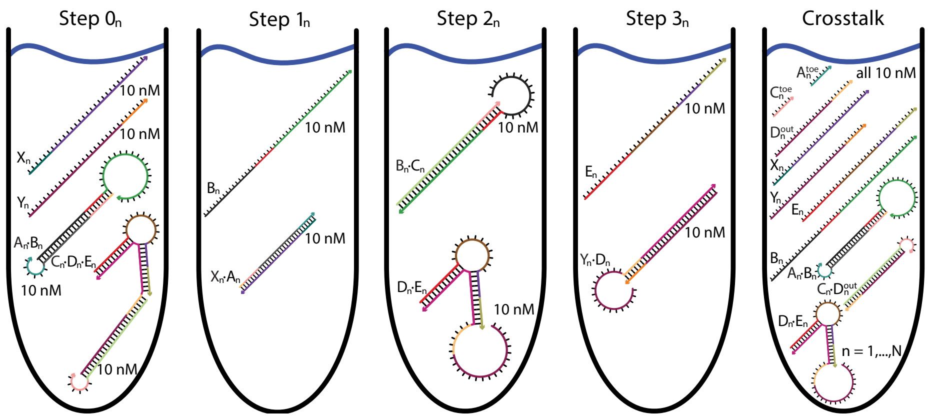
- $\lambda^{\text{simple}}_n \equiv \{\text{A}\cdot\text{B, C}\cdot\text{D}^{\text{out}}\text{, D}\cdot\text{E}\}_n$
- $\lambda^{\text{ss-out}}_n \equiv\{\text{X, Y, B, D}^{\text{out}}\text{, E}\}_n$
- $\lambda^{\text{ss-in}}_n \equiv \{\text{A}^{\text{toe}}\text{, C}^{\text{toe}}\text{, D}^{\text{toe}}\}_n$
- $\lambda^{\text{reactive}}_n \equiv \{\text{A}\cdot\text{B, C}\cdot\text{D}^{\text{out}}\text{, D}\cdot\text{E, X, Y, B, D}^{\text{out}}\text{, E, A}^{\text{toe}}\text{, C}^{\text{toe}}\}_n$
- $\lambda^{\text{cognate}}_n \equiv \{\text{X}\cdot\text{A, B}\cdot\text{C, Y}\cdot\text{D, X}\cdot\text{A}^{\text{toe}}\text{, B}\cdot\text{C}^{\text{toe}}\text{, Y}\cdot\text{D}^{\text{out}}\}_n$

based on the definitions (listed 5′ to 3′ using the sequence domain notation of Figure B.2):

- A ≡ h*-f*-e*-a*

- $\mathrm{A}^{\mathrm{toe}} \equiv$ a*
- B ≡ e-f-g
- C ≡ g*-f*
- $\mathrm{C}^{\mathrm{toe}} \equiv$ f*
- $\mathrm{D} \equiv \mathrm{D}^{\mathrm{out}}\text{-}\mathrm{D}^{\mathrm{in}}$
- $\mathrm{D}^{\mathrm{toe}} \equiv$ d*
- $\mathrm{D}^{\mathrm{in}} \equiv$ c*
- $\mathrm{D}^{\mathrm{out}} \equiv$ i-d*
- E ≡ w-x-y-z
- X ≡ a-b
- Y ≡ c-d

Note: $\mathrm{D}_n^{\mathrm{toe}}$ is contained in $\mathrm{D}_n^{\mathrm{out}}$, providing a toehold adjacent to $\mathrm{D}_n^{\mathrm{in}}$. To avoid redundancy, we omit $\mathrm{D}_n^{\mathrm{toe}}$ from $\lambda_n^{\mathrm{reactive}}$ because it is a subsequence of $\mathrm{D}_n^{\mathrm{out}}$ in $\lambda_n^{\mathrm{ss\text{-}out}}$.



Tube	On-targets (Ψ_h^{on})	Off-targets (Ψ_h^{off})
Step 0_n	$\{\text{X, Y, A}\cdot\text{B, C}\cdot\text{D}\cdot\text{E}\}_n$	$\{\text{A, B, C, D, E, C}\cdot\text{D, D}\cdot\text{E}\}_n \cup \Psi_{0_n}^{L\leq L_{\max}} - \{\text{X}\cdot\text{A}\}_n$
Step 1_n	$\{\text{X}\cdot\text{A, B}\}_n$	$\{\text{X, A}\cdot\text{B}\}_n \cup \Psi_{1_n}^{L\leq L_{\max}}$
Step 2_n	$\{\text{B}\cdot\text{C, D}\cdot\text{E}\}_n$	$\{\text{B, C}\cdot\text{D}\cdot\text{E}\}_n \cup \Psi_{2_n}^{L\leq L_{\max}}$
Step 3_n	$\{\text{Y}\cdot\text{D, E}\}_n$	$\{\text{Y, D}\cdot\text{E}\}_n \cup \Psi_{3_n}^{L\leq L_{\max}}$
Crosstalk	$\cup_{n=1,\ldots,N}\{\lambda_n^{\text{reactive}}\}$	$\Psi_{\text{global}}^{L\leq L_{\max}} - \cup_{n=1,\ldots,N}\{\lambda_n^{\text{cognate}}\}$

Figure B.7: Target test tubes for Boolean logic AND using toehold sequestration gates (reaction pathway of Figure B.2). Top: Target test tube schematics. Bottom: Target test tube details. Each target test tube contains the depicted on-target complexes (each with the depicted target structure and a target concentration of 10 nM) and the off-target complexes listed in the table (each with vanishing target concentration). To simultaneously design N orthogonal systems, the total number of target test tubes is $|\Omega| = 4N + 1$. $L_{\max} = 2$ for all tubes. Design conditions: DNA in 1 M Na$^+$ at 25 °C.

Self-Assembly of a 3-Arm Junction via CHA

Target test tubes are defined using the specification of Section 3.2.3 with the following definitions. The total number of target test tubes is $|\Omega| = \sum_{n=1,\ldots,N}$ {Step 0, Step 1, Step 2, Step 3a, Step 3b}$_n$ + Crosstalk = $5N + 1$; the target test tubes in the multistate test tube ensemble, Ω, are indexed by $h = 1, \ldots, 5N + 1$. $L_{\max} = 2$ for all tubes.

Reactants for system *n*

- Target: X_n
- Hairpins: $\{\mathrm{A}, \mathrm{B}, \mathrm{C}\}_n$

Elementary step tubes for system *n*

- Step 0_n: $\Psi^{\text{products}}_{0_n} \equiv \{\mathrm{X}, \mathrm{A}, \mathrm{B}, \mathrm{C}\}_n$; $\Psi^{\text{reactants}}_{0_n} \equiv \emptyset$; $\Psi^{\text{exclude}}_{0_n} \equiv \{\mathrm{X{\cdot}A}, \mathrm{X{\cdot}B}\}_n$
- Step 1_n: $\Psi^{\text{products}}_{1_n} \equiv \{\mathrm{X{\cdot}A}\}_n$; $\Psi^{\text{reactants}}_{1_n} \equiv \{\mathrm{X}, \mathrm{A}\}_n$; $\Psi^{\text{exclude}}_{1_n} \equiv \emptyset$
- Step 2_n: $\Psi^{\text{products}}_{2_n} \equiv \{\mathrm{X{\cdot}A{\cdot}B}\}_n$; $\Psi^{\text{reactants}}_{2_n} \equiv \{\mathrm{X{\cdot}A}, \mathrm{B}\}_n$; $\Psi^{\text{exclude}}_{2_n} \equiv \emptyset$
- Step $3\mathrm{a}_n$: $\Psi^{\text{products}}_{3\mathrm{a}_n} \equiv \{\mathrm{X{\cdot}A{\cdot}B{\cdot}C}\}_n$; $\Psi^{\text{reactants}}_{3\mathrm{a}_n} \equiv \{\mathrm{X{\cdot}A{\cdot}B}, \mathrm{C}\}_n$; $\Psi^{\text{exclude}}_{3\mathrm{a}_n} \equiv \emptyset$
- Step $3\mathrm{b}_n$: $\Psi^{\text{products}}_{3\mathrm{b}_n} \equiv \{\mathrm{X}, \mathrm{A{\cdot}B{\cdot}C}\}_n$; $\Psi^{\text{reactants}}_{3\mathrm{b}_n} \equiv \{\mathrm{X{\cdot}A{\cdot}B{\cdot}C}\}_n$; $\Psi^{\text{exclude}}_{3\mathrm{b}_n} \equiv \emptyset$

Note: Step 3 combining an assembly operation (Step 3a; addition of C) with a disassembly operation (Step 3b; removal of X) is described using two target test tubes; the Step 3a tube prevents completion of the full operation by excluding the final product A·B·C from the ensemble ($L_{\max} = 2$ includes all off-targets up to dimers).

Crosstalk tube

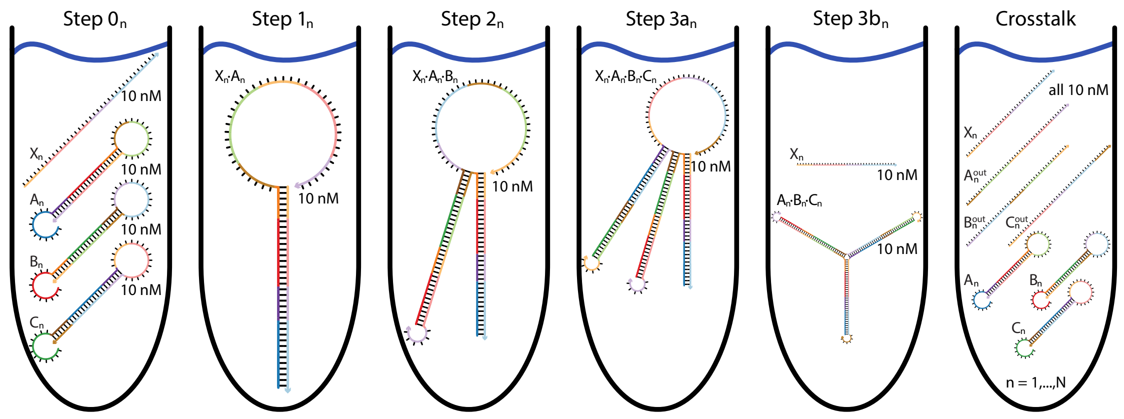
- Crosstalk tube: $\Psi^{\text{reactive}}_{\text{global}} \equiv \cup_{n=1,\ldots,N}\{\lambda^{\text{reactive}}_n\}$; $\Psi^{\text{crosstalk}}_{\text{global}} \equiv \Psi^{L \leq L_{\max}}_{\text{global}} - \cup_{n=1,\ldots,N}\{\lambda^{\text{cognate}}_n\}$

The reactive species and cognate products for system *n* are:

- $\lambda^{\text{simple}}_n \equiv \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}_n$
- $\lambda^{\text{ss-out}}_n \equiv \{\mathrm{X}, \mathrm{A}^{\text{out}}, \mathrm{B}^{\text{out}}, \mathrm{C}^{\text{out}}\}_n$
- $\lambda^{\text{ss-in}}_n \equiv \{\mathrm{A}^{\text{toe}}, \mathrm{B}^{\text{toe}}, \mathrm{C}^{\text{toe}}\}_n$
- $\lambda^{\text{reactive}}_n \equiv \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{X}, \mathrm{A}^{\text{out}}, \mathrm{B}^{\text{out}}, \mathrm{C}^{\text{out}}\}_n$
- $\lambda^{\text{cognate}}_n \equiv \{\mathrm{X{\cdot}A}, \mathrm{X{\cdot}B}, \mathrm{A}^{\text{out}}{\cdot}\mathrm{B}, \mathrm{B}^{\text{out}}{\cdot}\mathrm{C}, \mathrm{C}^{\text{out}}{\cdot}\mathrm{A}, \mathrm{C}^{\text{out}}{\cdot}\mathrm{B}, \mathrm{B}^{\text{out}}{\cdot}\mathrm{A}, \mathrm{A}^{\text{out}}{\cdot}\mathrm{C}\}_n$

based on the definitions (listed 5′ to 3′ using the sequence domain notation of Figure B.3):

- A ≡ A^{in}-A^{out}
- A^{toe} ≡ a
- A^{in} ≡ a-x-b-y
- A^{out} ≡ z*-c*-y*-b*-x*
- B ≡ B^{in}-B^{out}
- B^{toe} ≡ b
- B^{in} ≡ b-y-c-z
- B^{out} ≡ x*-a*-z*-c*-y*
- C ≡ C^{in}-C^{out}
- C^{toe} ≡ c
- C^{in} ≡ c-z-a-x
- C^{out} ≡ y*-b*-x*-a*-z*
- X ≡ y*-b*-x*-a*



Tube	On-targets (Ψ_h^{on})	Off-targets (Ψ_h^{off})
Step 0_n	$\{\text{X, A, B, C}\}_n$	$\Psi_{0_n}^{L\leq L_{\max}} - \{\text{X}\cdot\text{A, X}\cdot\text{B}\}_n$
Step 1_n	$\{\text{X}\cdot\text{A}\}_n$	$\{\text{X, A}\}_n \cup \Psi_{1_n}^{L\leq L_{\max}}$
Step 2_n	$\{\text{X}\cdot\text{A}\cdot\text{B}\}_n$	$\{\text{X}\cdot\text{A, B}\}_n \cup \Psi_{2_n}^{L\leq L_{\max}}$
Step 3a_n	$\{\text{X}\cdot\text{A}\cdot\text{B}\cdot\text{C}\}_n$	$\{\text{X}\cdot\text{A}\cdot\text{B, C}\}_n \cup \Psi_{3\text{a}_n}^{L\leq L_{\max}}$
Step 3b_n	$\{\text{X, A}\cdot\text{B}\cdot\text{C}\}_n$	$\{\text{X}\cdot\text{A}\cdot\text{B}\cdot\text{C}\}_n \cup \Psi_{3\text{b}_n}^{L\leq L_{\max}}$
Crosstalk	$\cup_{n=1,\dots,N}\{\lambda_n^{\text{reactive}}\}$	$\Psi_{\text{global}}^{L\leq L_{\max}} - \cup_{n=1,\dots,N}\{\lambda_n^{\text{cognate}}\}$

Figure B.8: Target test tubes for self-assembly of a 3-arm junction via CHA (reaction pathway of Figure B.3). Top: Target test tube schematics. Bottom: Target test tube details. Each target test tube contains the depicted on-target complexes (each with the depicted target structure and a target concentration of 10 nM) and the off-target complexes listed in the table (each with vanishing target concentration). To simultaneously design N orthogonal systems, the total number of target test tubes is $|\Omega| = 5N + 1$. $L_{\max} = 2$ for all tubes. Design conditions: DNA in 1 M Na$^+$ at 25 °C.

Boolean Logic AND using a Cooperative Hybridization Gate

Target test tubes are defined using the specification of Section 3.2.3 with the following definitions. The total number of target test tubes is $|\Omega| = \sum_{n=1,\ldots,N}$ {Step 0, Step 1}$_n$ + Crosstalk = $2N + 1$; the target test tubes in the multistate test tube ensemble, Ω, are indexed by $h = 1, \ldots, 2N + 1$. $L_{\max} = 2$ for all tubes.

Reactants for system n

- Targets: $\{\text{X, Y}\}_n$
- Cooperative gate: $\{\text{A·B}\}_n$

Elementary step tubes for system n

- Step 0_n: $\Psi_{0_n}^{\text{products}} \equiv \{\text{X, Y, A·B}\}_n$; $\Psi_{0_n}^{\text{reactants}} \equiv \{\text{A, B}\}_n$; $\Psi_{0_n}^{\text{exclude}} \equiv \emptyset$
- Step 1_n: $\Psi_{1_n}^{\text{products}} \equiv \{\text{Y·A·X, B}\}_n$; $\Psi_{1_n}^{\text{reactants}} \equiv \{\text{X, Y, A·B}\}_n$; $\Psi_{1_n}^{\text{exclude}} \equiv \emptyset$

Note: In the Step 0_n tube, the reactants are prevented from generating the product Y·A·X, because this trimer is excluded from the test tube ensemble ($L_{\max} = 2$ includes all off-targets up to dimers).

Crosstalk tube

- Crosstalk tube: $\Psi_{\text{global}}^{\text{reactive}} \equiv \cup_{n=1,\ldots,N}\{\lambda_n^{\text{reactive}}\}$; $\Psi_{\text{global}}^{\text{crosstalk}} \equiv \Psi_{\text{global}}^{L \leq L_{\max}} - \cup_{n=1,\ldots,N}\{\lambda_n^{\text{cognate}}\}$

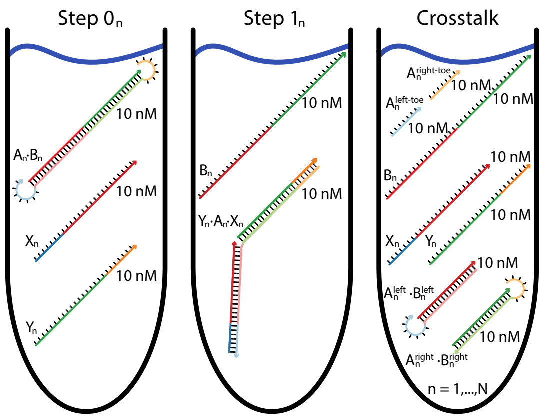
The reactive species and cognate products for system n are:

- $\lambda_n^{\text{simple}} \equiv \{\text{A}^{\text{left}}\text{·B}^{\text{left}}, \text{A}^{\text{right}}\text{·B}^{\text{right}}\}_n$
- $\lambda_n^{\text{ss-out}} \equiv \{\text{X, Y, B}\}_n$
- $\lambda_n^{\text{ss-in}} \equiv \{\text{A}^{\text{left-toe}}, \text{A}^{\text{right-toe}}\}_n$
- $\lambda_n^{\text{reactive}} \equiv \{\text{A}^{\text{left}}\text{·B}^{\text{left}}, \text{A}^{\text{right}}\text{·B}^{\text{right}}, \text{X, Y, B, A}^{\text{left-toe}}, \text{A}^{\text{right-toe}}\}_n$
- $\lambda_n^{\text{cognate}} \equiv \{\text{X·A}^{\text{left}}, \text{Y·A}^{\text{right}}, \text{X·A}^{\text{left-toe}}, \text{Y·A}^{\text{right-toe}}, \text{A}^{\text{left}}\text{·B}, \text{A}^{\text{right}}\text{·B}\}_n$

based on the definitions (listed 5′ to 3′ using the sequence domain notation of Figure B.4):

- $\text{A} \equiv \text{A}^{\text{right}}\text{-A}^{\text{left}}$
- $\text{A}^{\text{left-toe}} \equiv \text{a*}$
- $\text{A}^{\text{left}} \equiv \text{b*-a*}$

- $A^{\text{right-toe}} \equiv d^*$
- $A^{\text{right}} \equiv d^*-c^*$
- $B \equiv B^{\text{left}}-B^{\text{right}}$
- $B^{\text{left}} \equiv b$
- $B^{\text{right}} \equiv c$
- $X \equiv a-b$
- $Y \equiv c-d$



Tube	On-targets (Ψ_h^{on})	Off-targets (Ψ_h^{off})
Step 0_n	$\{X, Y, A{\cdot}B\}_n$	$\{A, B\}_n \cup \Psi_{0_n}^{L \le L_{\max}}$
Step 1_n	$\{Y{\cdot}A{\cdot}X, B\}_n$	$\{X, Y, A{\cdot}B\}_n \cup \Psi_{1_n}^{L \le L_{\max}}$
Crosstalk	$\cup_{n=1,\ldots,N}\{\lambda_n^{\text{reactive}}\}$	$\Psi_{\text{global}}^{L \le L_{\max}} - \cup_{n=1,\ldots,N}\{\lambda_n^{\text{cognate}}\}$

Figure B.9: Target test tubes for Boolean logic AND using a cooperative hybridization gate (reaction pathway of Figure B.4). Top: Target test tube schematics. Bottom: Target test tube details. Each target test tube contains the depicted on-target complexes (each with the depicted target structure and a target concentration of 10 nM) and the off-target complexes listed in the table (each with vanishing target concentration). To simultaneously design N orthogonal systems, the total number of target test tubes is $|\Omega| = 2N + 1$. $L_{\max} = 2$ for all tubes. Design conditions: DNA in 1 M Na$^+$ at 25 °C.

Conditional Dicer Substrate Formation via Shape and Sequence Transduction with scRNAs

Target test tubes are defined using the specification of Section 3.2.3 with the following definitions. The total number of target test tubes is $|\Omega| = \sum_{n=1,\ldots,N}$ $\{\text{Step 0, Step 1, Step 2}\}_n$ + Crosstalk = $3N + 1$; the target test tubes in the multistate test tube ensemble, Ω, are indexed by $h = 1, \ldots, 3N+1$. $L_{\max} = 2$ for all tubes.

Reactants for system *n*

- Target: X_n
- scRNAs: $\{A{\cdot}B, C\}_n$

Elementary step tubes for system *n*

- Step 0_n: $\Psi^{\text{products}}_{0_n} \equiv \{X, A{\cdot}B, C\}_n$; $\Psi^{\text{reactants}}_{0_n} \equiv \{A, B{\cdot}C\}_n$; $\Psi^{\text{exclude}}_{0_n} \equiv \{X{\cdot}A\}$
- Step 1_n: $\Psi^{\text{products}}_{1_n} \equiv \{X{\cdot}A, B\}_n$; $\Psi^{\text{reactants}}_{1_n} \equiv \{X, A{\cdot}B\}_n$; $\Psi^{\text{exclude}}_{1_n} \equiv \emptyset$
- Step 2_n: $\Psi^{\text{products}}_{2_n} \equiv \{B{\cdot}C\}_n$; $\Psi^{\text{reactants}}_{2_n} \equiv \{B, C\}_n$; $\Psi^{\text{exclude}}_{2_n} \equiv \emptyset$

Crosstalk tube

- Crosstalk tube: $\Psi^{\text{reactive}}_{\text{global}} \equiv \cup_{n=1,\ldots,N}\{\lambda^{\text{reactive}}_n\}$; $\Psi^{\text{crosstalk}}_{\text{global}} \equiv \Psi^{L \le L_{\max}}_{\text{global}} - \cup_{n=1,\ldots,N}\{\lambda^{\text{cognate}}_n\}$

The reactive species and cognate products for system *n* are:

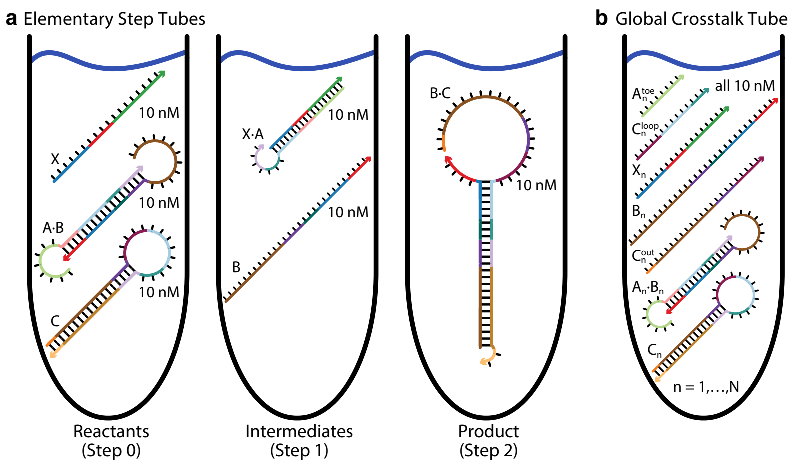
- $\lambda^{\text{simple}}_n = \{A{\cdot}B, C\}_n$
- $\lambda^{\text{ss-out}}_n = \{X, B, C^{\text{out}}\}_n$
- $\lambda^{\text{ss-in}}_n = \{A^{\text{toe}}, C^{\text{loop}}\}_n$
- $\lambda^{\text{reactive}}_n = \{A{\cdot}B, C, X, B, C^{\text{out}}, A^{\text{toe}}, C^{\text{loop}}\}_n$
- $\lambda^{\text{cognate}}_n = \{X{\cdot}A, B{\cdot}C, X{\cdot}A^{\text{toe}}, B{\cdot}C^{\text{loop}}\}_n$

based on the definitions (listed 5′ to 3′ using the sequence domain notation of Figure B.5):

- A ≡ c*-b*-a*-z*-y*
- A^{toe} ≡ c*
- B ≡ x-y-z-a-b

- $\mathrm{C} \equiv \mathrm{C}^{\mathrm{out}}$-$\mathrm{C}^{\mathrm{in}}$
- $\mathrm{C}^{\mathrm{loop}} \equiv$ s-a*-z*
- $\mathrm{C}^{\mathrm{in}} \equiv$ a*-z*-y*-x*-w*
- $\mathrm{C}^{\mathrm{out}} \equiv$ w-x-y-s
- $\mathrm{X} \equiv$ a-b-c

Note: $\mathrm{C}_n^{\mathrm{loop}}$ includes portions of both $\mathrm{C}_n^{\mathrm{in}}$ and $\mathrm{C}_n^{\mathrm{out}}$. Including $\mathrm{C}_n^{\mathrm{loop}}$ in $\lambda_n^{\mathrm{reactive}}$ is not redundant with inclusion of C_n because pairing to the loop would cause a pseudoknot and hence will not be checked by the ensemble except if the interaction opens the hairpin. We want to be able to check nucleation with the loop even when the hairpin remains closed, so we include $\mathrm{C}_n^{\mathrm{loop}}$ in $\lambda_n^{\mathrm{reactive}}$.



Tube	On-targets (Ψ_h^{on})	Off-targets (Ψ_h^{off})
Step 0_n	$\{\text{X, A·B, C}\}_n$	$\{\text{A, B·C}\}_n \cup \Psi_{0_n}^{L\le L_{\max}} - \{\text{X·A}\}$
Step 1_n	$\{\text{X·A, B}\}_n$	$\{\text{X, A·B}\}_n \cup \Psi_{1_n}^{L\le L_{\max}}$
Step 2_n	$\{\text{B·C}\}_n$	$\{\text{B, C}\}_n \cup \Psi_{2_n}^{L\le L_{\max}}$
Crosstalk	$\cup_{n=1,\ldots,N}\{\lambda_n^{\text{reactive}}\}$	$\Psi_{\text{global}}^{L\le L_{\max}} - \cup_{n=1,\ldots,N}\{\lambda_n^{\text{cognate}}\}$

Figure B.10: Target test tubes for conditional Dicer substrate formation via shape and sequence transduction with scRNAs (reaction pathway of Figure B.5). Top: Target test tube schematics. Bottom: Target test tube details. Each target test tube contains the depicted on-target complexes (each with the depicted target structure and a target concentration of 10 nM) and the off-target complexes listed in the table (each with vanishing target concentration). To simultaneously design N orthogonal systems, the total number of target test tubes is $|\Omega| = 3N + 1$. $L_{\max} = 2$ for all tubes. Design conditions: RNA in 1 M Na$^+$ at 37 °C.

B.2.2 Algorithm Performance

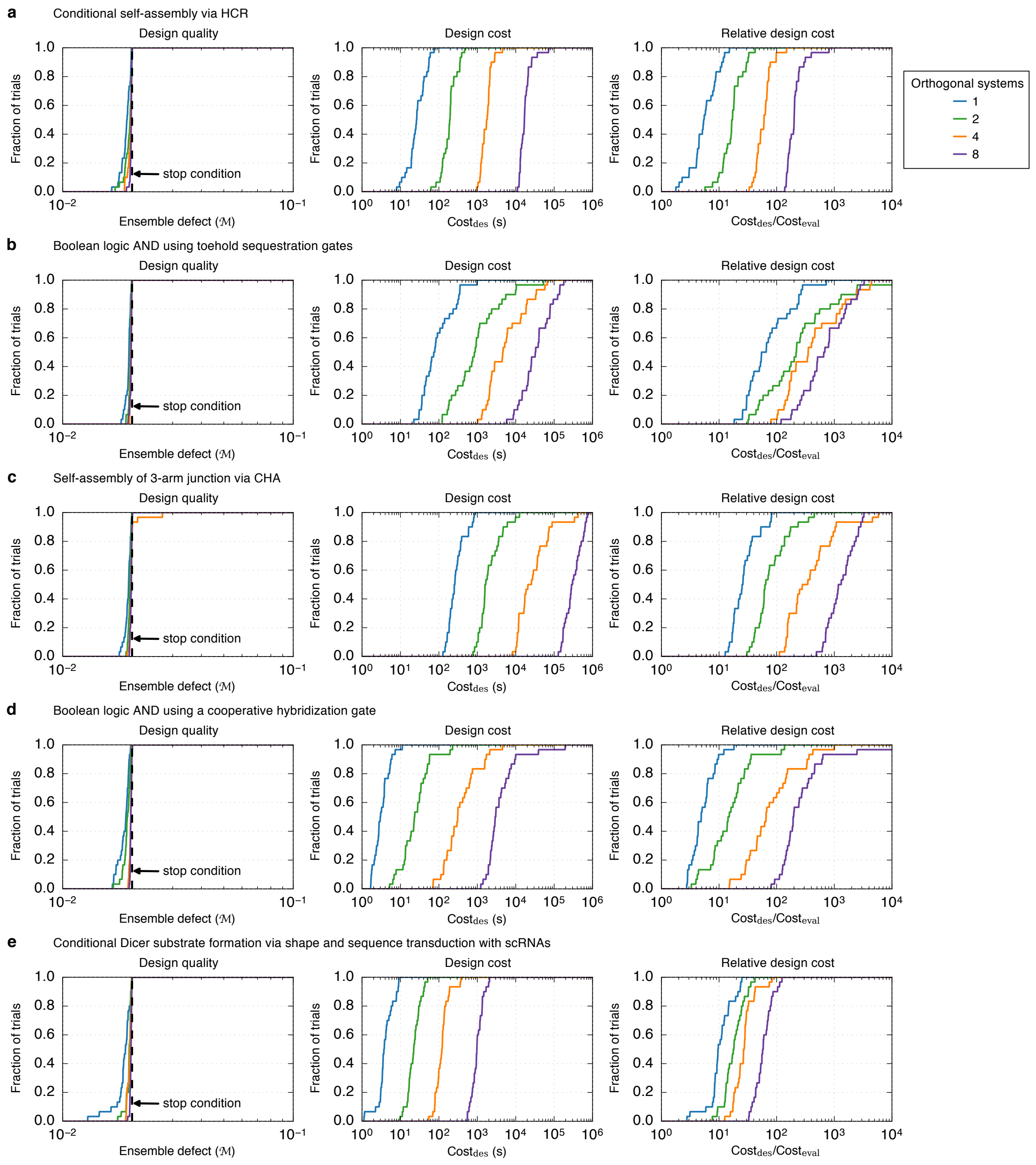


Figure B.11: Algorithm performance for design of 1, 2, 4, or 8 orthogonal systems based on the target test tubes of Section 3.2.3. Left: Design quality. The stop condition is depicted as a dashed black line. Middle: Design cost. Right: Cost of sequence design relative to a single evaluation of the objective function. (a) Conditional self-assembly via HCR. (b) Boolean logic AND using toehold sequestration gates. (c) Self-assembly of 3-arm junction via CHA. (d) Boolean logic AND using a cooperative hybridization gate. (e) Conditional Dicer substrate formation via shape and sequence transduction with scRNAs.

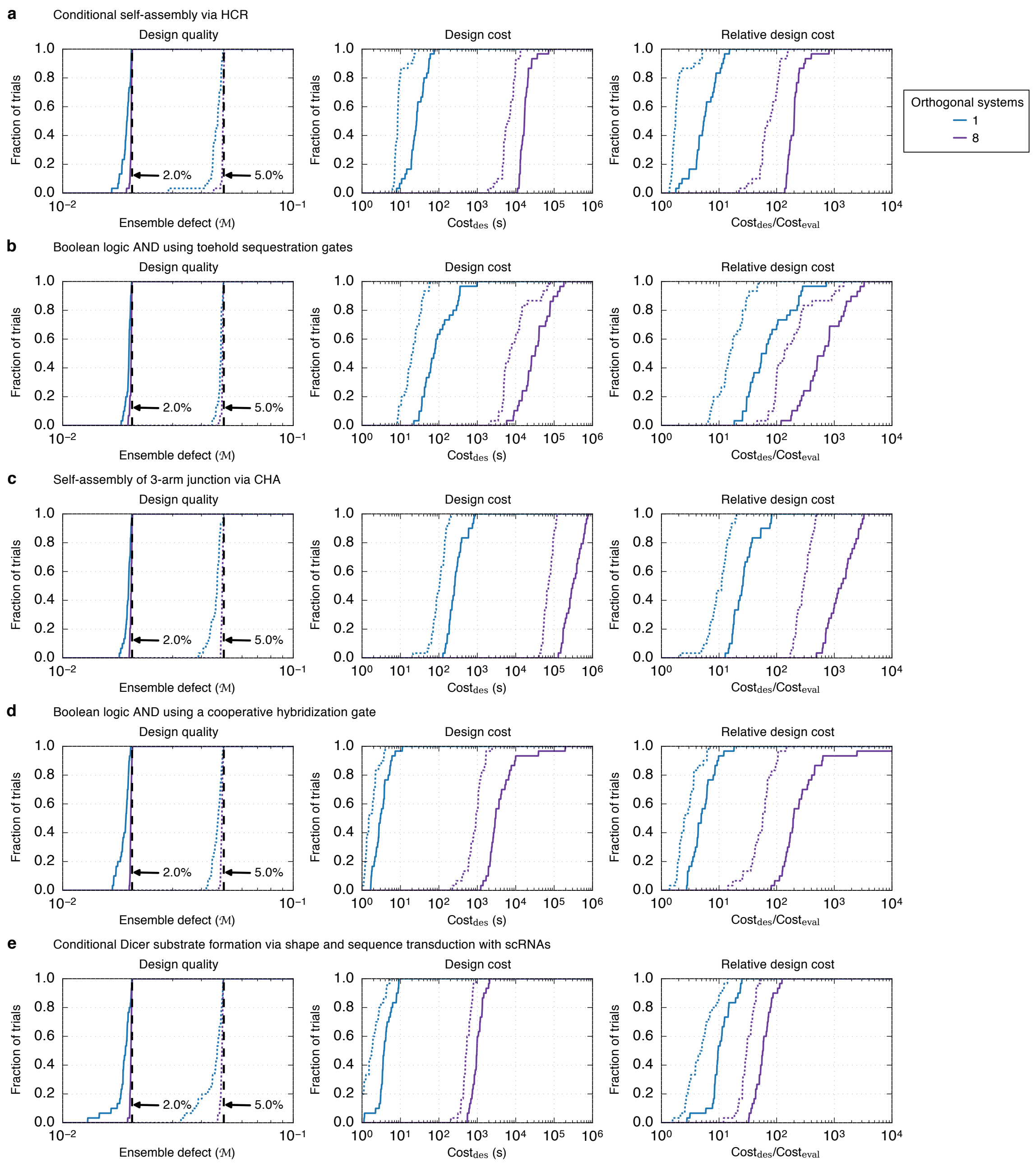


Figure B.12: Reduced design cost and quality using $f_{\text{stop}} = 0.05$ (dotted lines) instead of $f_{\text{stop}} = 0.02$ (solid lines) for design of 1 or 8 orthogonal systems based on the target test tubes of Section 3.2.3. Left: Design quality. The stop conditions are depicted as dashed black lines. Middle: Design cost. Right: Cost of sequence design relative to a single evaluation of the objective function. (a) Conditional self-assembly via HCR. (b) Boolean logic AND using toehold sequestration gates. (c) Self-assembly of 3-arm junction via CHA. (d) Boolean logic AND using a cooperative hybridization gate. (e) Conditional Dicer substrate formation via shape and sequence transduction with scRNAs.

B.2.3 Residual Defects

For each case study, the residual defect plots that follow display for each target test tube $h \in \Omega$:

- The structural defect and concentration defect contributions of each on-target complex to the test tube ensemble defect, $\mathcal{M}_h$. The fact that bars are not depicted for off-targets does not mean that the defects associated with off-targets are neglected. By conservation of mass, nonzero off-target concentrations imply deficiencies in on-target concentrations, and these on-target concentration defects are depicted in the bar graphs and incorporated in $\mathcal{M}_h$ via (3.1) and (3.5).
- The total structural defect and total concentration defect contributions of the tube to the multistate test tube ensemble defect, $\mathcal{M}$.

Conditional Self-Assembly via HCR

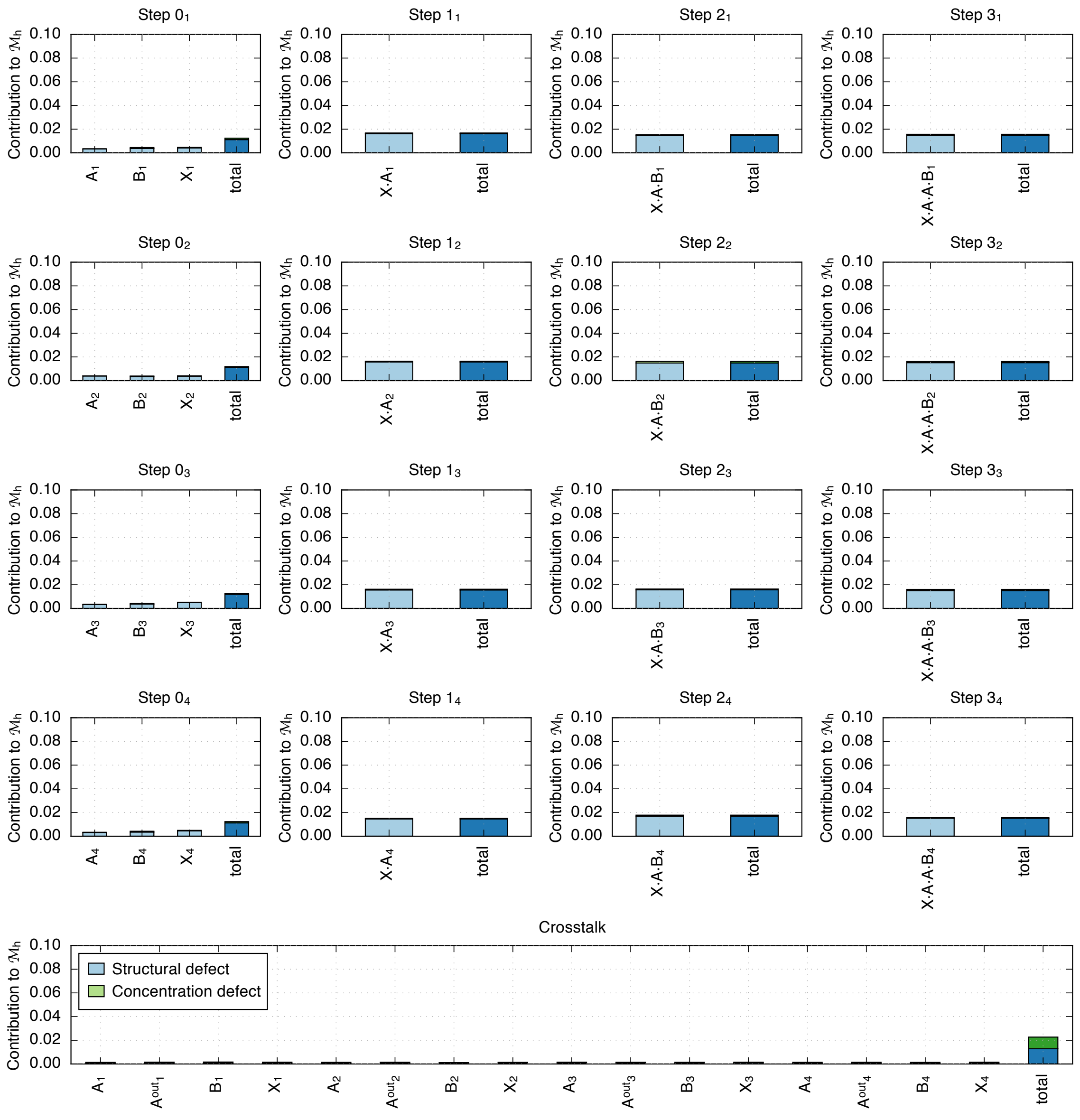


Figure B.13: Residual defects for conditional self-assembly via HCR (N = 4 orthogonal systems for target test tubes of Figure B.6). Each panel corresponds to a different tube $h \in \{1, \dots, 4N + 1\}$. For each tube h, the structural defect and concentration defect contributions to the tube ensemble defect, $\mathcal{M}_h$, are depicted for each complex (pale shaded bars). The total structural defect and total concentration defect contributions to the multistate test tube ensemble defect, $\mathcal{M}$, are also depicted for each tube (dark shaded bars). Each bar represents the mean over 30 independent design trials with stop condition $\mathcal{M} \leq 0.02$. All nucleotide, complex, and tube weights are left at the default value of 1 except for the global crosstalk tube which is assigned a weight of N to prevent the effect of crosstalk from being diluted in the design objective function as the number of orthogonal systems increases.

Boolean Logic AND using Toehold Sequestration Gates

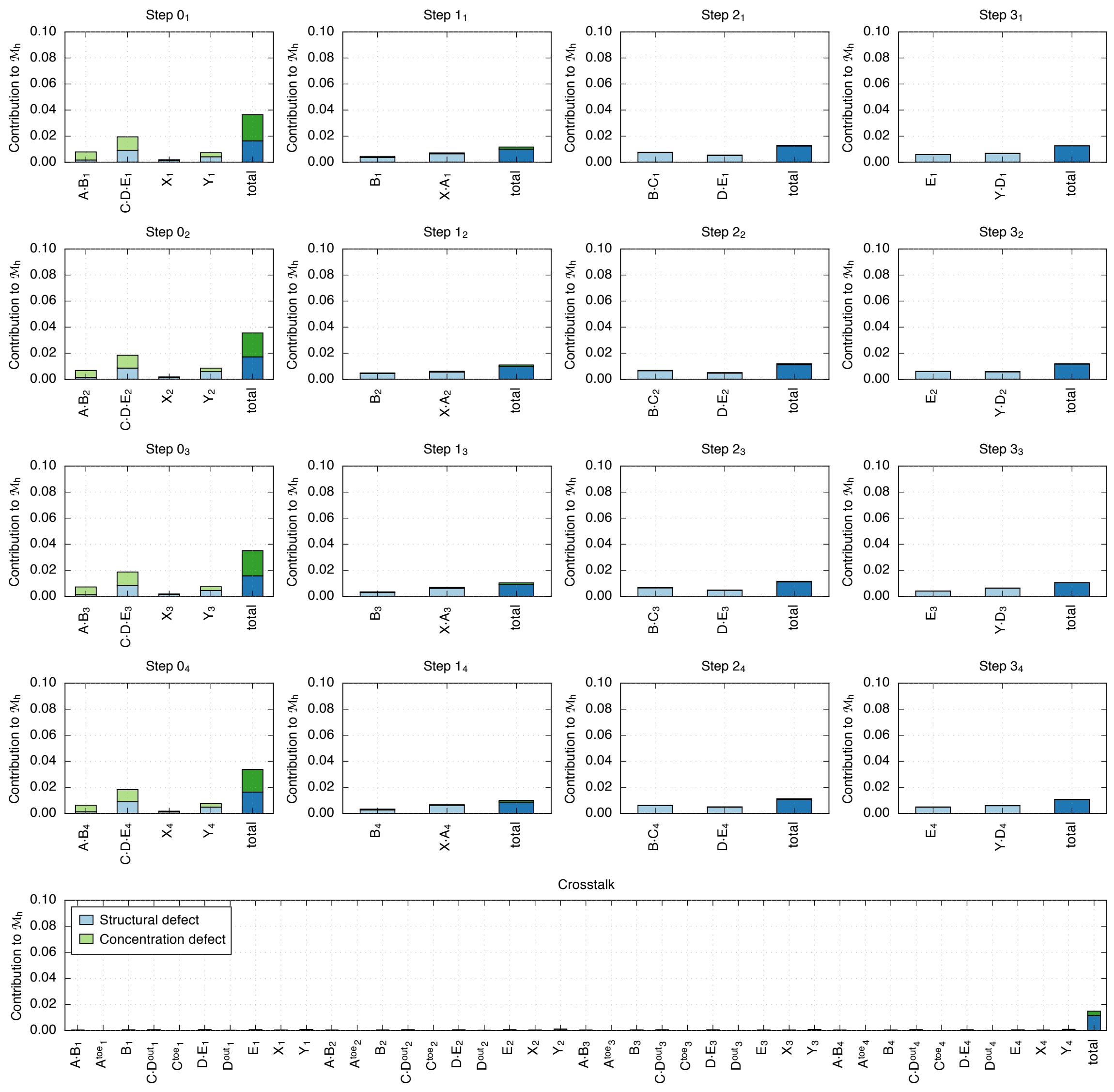


Figure B.14: Residual defects for Boolean logic AND using toehold sequestration gates ($N = 4$ orthogonal systems for target test tubes of Figure B.7). Each panel corresponds to a different tube $h \in \{1, \ldots, 4N + 1\}$. For each tube h, the structural defect and concentration defect contributions to the tube ensemble defect, $\mathcal{M}_h$, are depicted for each complex (pale shaded bars). The total structural defect and total concentration defect contributions to the multistate test tube ensemble defect, $\mathcal{M}$, are also depicted for each tube (dark shaded bars). Each bar represents the mean over 30 independent design trials with stop condition $\mathcal{M} \leq 0.02$. All nucleotide, complex, and tube weights are left at the default value of 1 except for the global crosstalk tube which is assigned a weight of N to prevent the effect of crosstalk from being diluted in the design objective function as the number of orthogonal systems increases.

Self-Assembly of a 3-Arm Junction via CHA

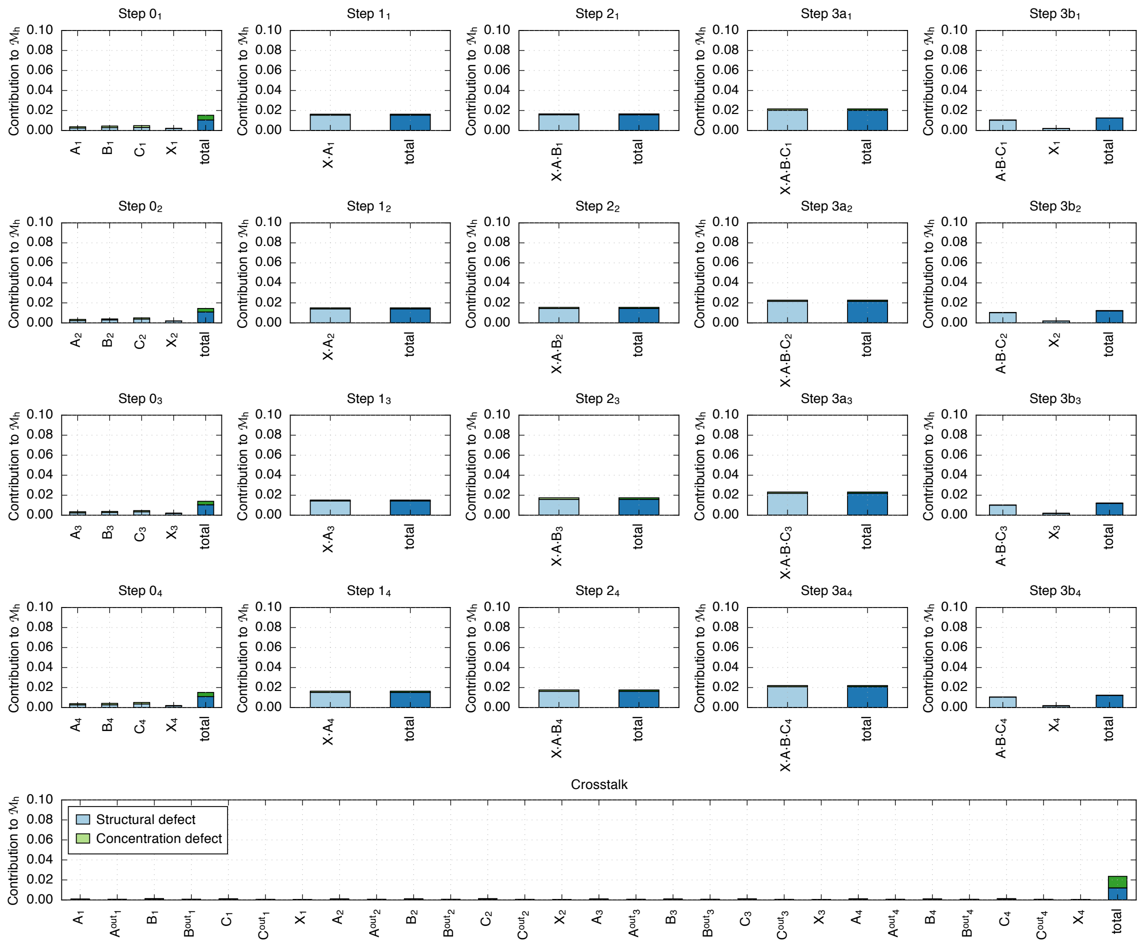


Figure B.15: Residual defects for self-assembly of a 3-arm junction via CHA ($N = 4$ orthogonal systems for target test tubes of Figure B.8). Each panel corresponds to a different tube $h \in \{1, \ldots, 5N + 1\}$. For each tube h, the structural defect and concentration defect contributions to the tube ensemble defect, $\mathcal{M}_h$, are depicted for each complex (pale shaded bars). The total structural defect and total concentration defect contributions to the multistate test tube ensemble defect, $\mathcal{M}$, are also depicted for each tube (dark shaded bars). Each bar represents the mean over 30 independent design trials with stop condition $\mathcal{M} \leq 0.02$. All nucleotide, complex, and tube weights are left at the default value of 1 except for the global crosstalk tube which is assigned a weight of N to prevent the effect of crosstalk from being diluted in the design objective function as the number of orthogonal systems increases.

Boolean Logic AND using a Cooperative Hybridization Gate

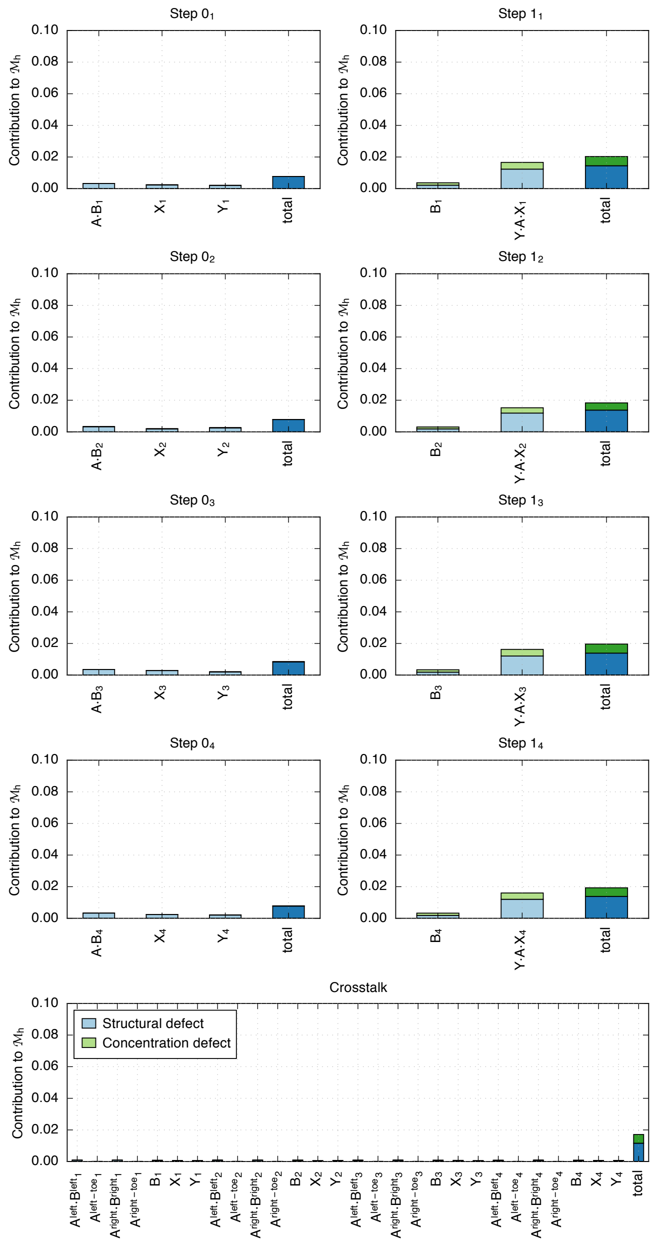


Figure B.16: Residual defects for Boolean logic AND using a cooperative hybridization gate ($N = 4$ orthogonal systems for target test tubes of Figure B.9). Each panel corresponds to a different tube $h \in \{1, \ldots, 2N + 1\}$. For each tube h, the structural defect and concentration defect contributions to the tube ensemble defect, $\mathcal{M}_h$, are depicted for each complex (pale shaded bars). The total structural defect and total concentration defect contributions to the multistate test tube ensemble defect, $\mathcal{M}$, are also depicted for each tube (dark shaded bars). Each bar represents the mean over 30 independent design trials with stop condition $\mathcal{M} \leq 0.02$. All nucleotide, complex, and tube weights are left at the default value of 1 except for the global crosstalk tube which is assigned a weight of N to prevent the effect of crosstalk from being diluted in the design objective function as the number of orthogonal systems increases.

Conditional Dicer Substrate Formation via Shape and Sequence Transduction with scRNAs

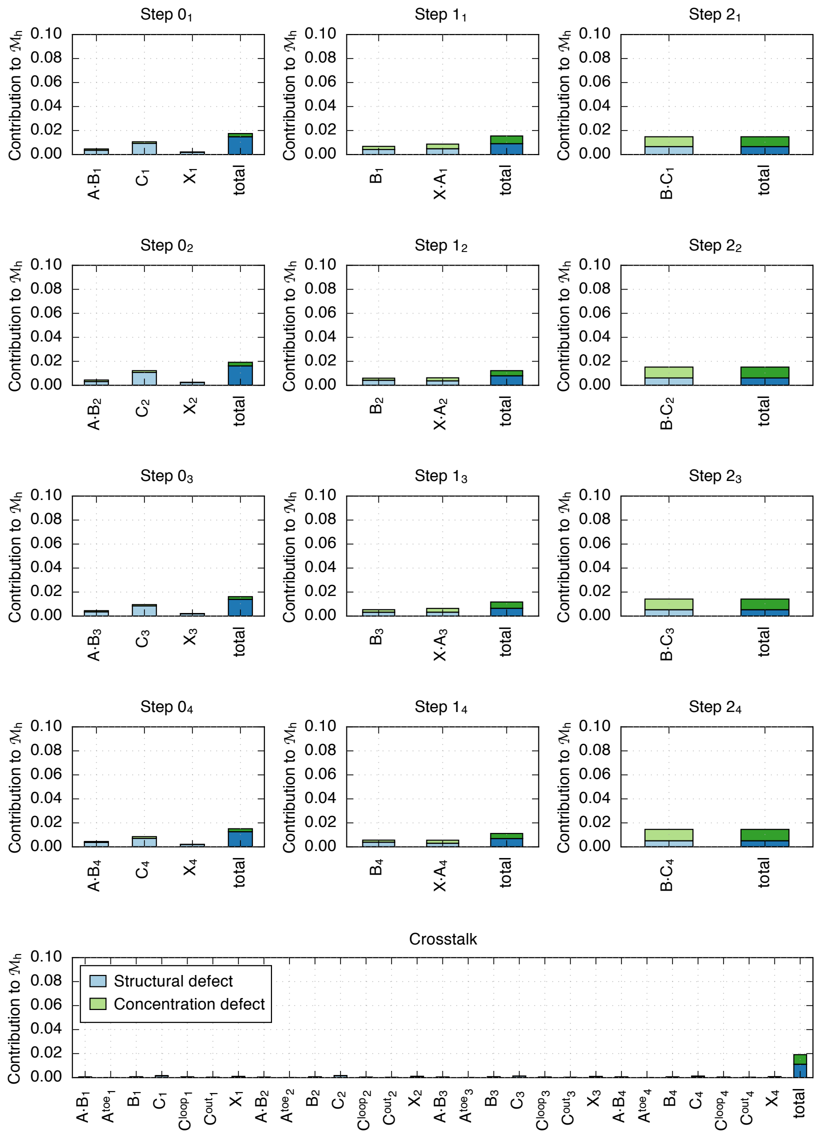


Figure B.17: Residual defects for conditional Dicer substrate formation via shape and sequence transduction with scRNAs ($N = 4$ orthogonal systems for target test tubes of Figure B.10). Each panel corresponds to a different tube $h \in \{1, \ldots, 3N + 1\}$. For each tube h, the structural defect and concentration defect contributions to the tube ensemble defect, $\mathcal{M}_h$, are depicted for each complex (pale shaded bars). The total structural defect and total concentration defect contributions to the multistate test tube ensemble defect, $\mathcal{M}$, are also depicted for each tube (dark shaded bars). Each bar represents the mean over 30 independent design trials with stop condition $\mathcal{M} \leq 0.02$. All nucleotide, complex, and tube weights are left at the default value of 1 except for the global crosstalk tube which is assigned a weight of N to prevent the effect of crosstalk from being diluted in the design objective function as the number of orthogonal systems increases.

B.2.4 Importance of Negative Design in Reducing Crosstalk

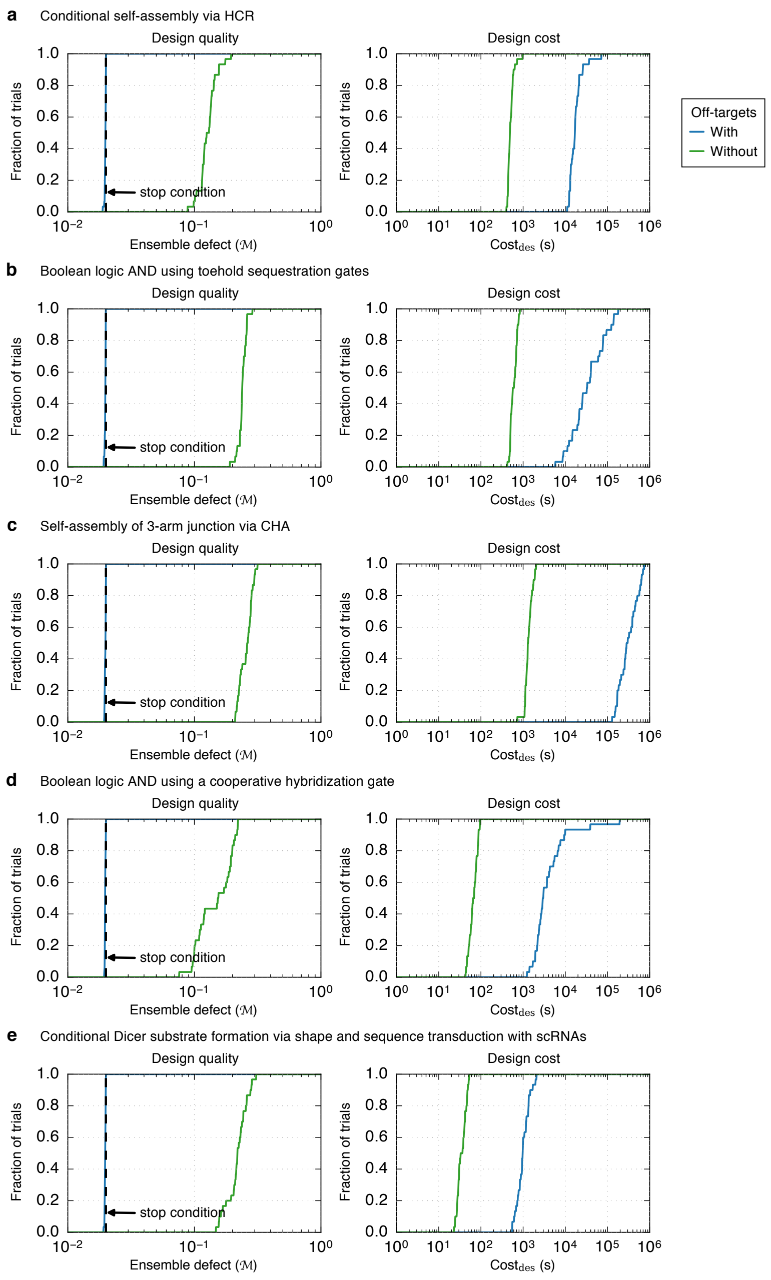


Figure B.18: Importance of negative design in reducing crosstalk ($N = 8$ orthogonal systems). Comparison of designs performed with or without off-targets in the design ensemble. Left: Design quality evaluated by calculating the multistate test tube ensemble defect ($\mathcal{M}$) over the ensemble containing off-targets. The stop condition is depicted as a dashed black line. Right: Design cost. (a) Conditional self-assembly via HCR. (b) Boolean logic AND using toehold sequestration gates. (c) Self-assembly of 3-arm junction via CHA. (d) Boolean logic AND using a cooperative hybridization gate. (e) Conditional Dicer substrate formation via shape and sequence transduction with scRNAs.

B.2.5 Effect of Sequence Constraints

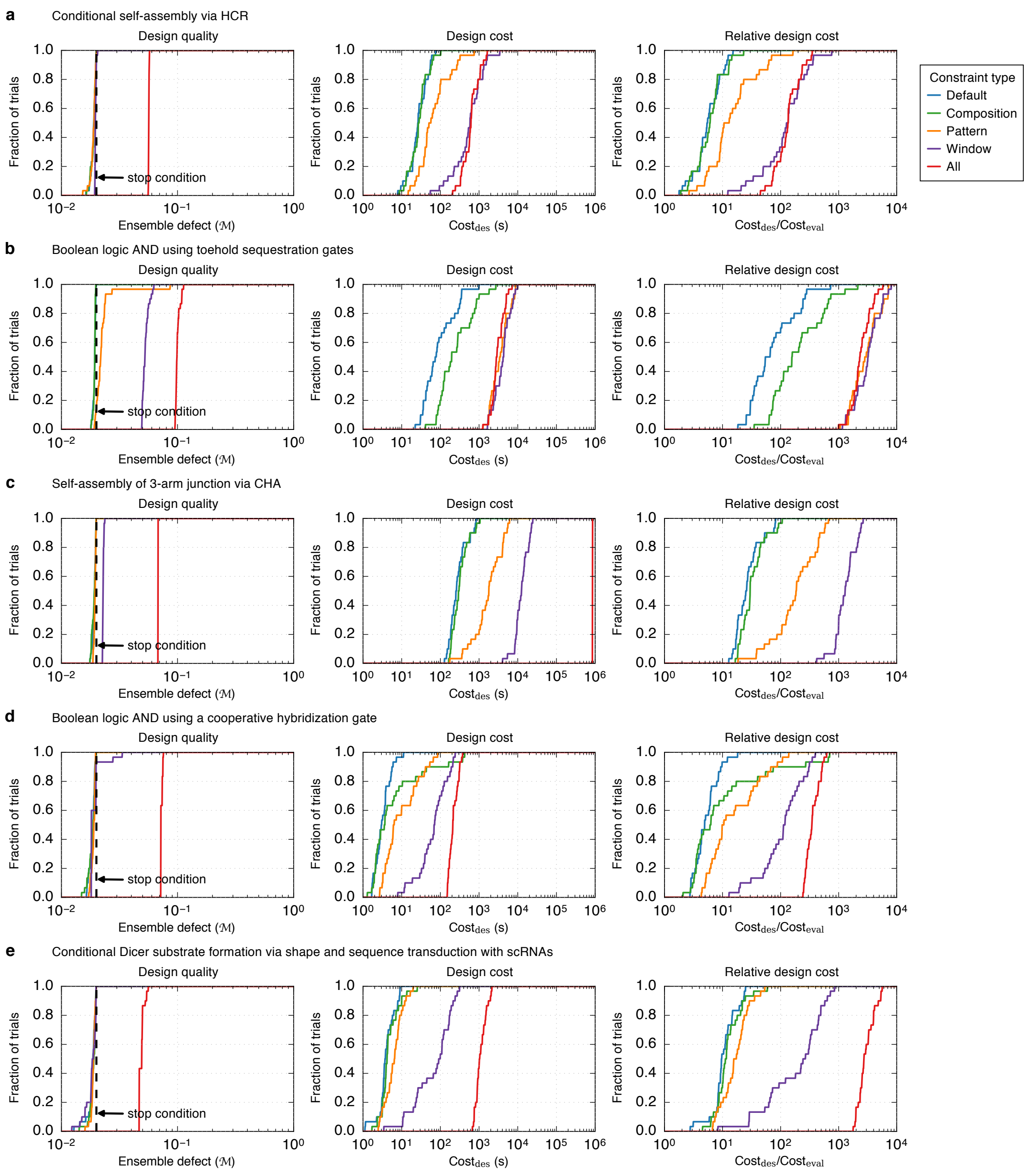


Figure B.19: Algorithm performance including explicit sequence constraints ($N = 1$ system). Default: implicit sequence constraints inherent to the reaction pathway (these constraints are also present in the other cases that follow). Composition constraint: fraction of S $\in [0.45, 0.55]$. Pattern constraint: prevent {`AAAA`, `CCCC`, `GGGG`, `UUUU`, `KKKKKK`, `MMMMMM`, `RRRRRR`, `SSSSSS`, `WWWWWW`, `YYYYYY`}. Window constraints: targets X and Y constrained to be subsequences of two different mRNAs (i.e., biological sequence constraints; see Section B.2.5). All: all of the above constraints. Left: Design quality. The stop condition is depicted as a dashed black line. Middle: Design cost. Right: Cost of sequence design relative to a single evaluation of the objective function. (a) Conditional self-assembly via HCR. (b) Boolean logic AND using toehold sequestration gates. (c) Self-assembly of 3-arm junction via CHA. (d) Boolean logic AND using a cooperative hybridization gate. (e) Conditional Dicer substrate formation via shape and sequence transduction with scRNAs. Note: for panel (c) with "All constraints", only 1 out of 30 independent design trials terminates within the depicted time frame.

mRNA Sequences used for Window Constraints

The window constraints used for the studies of Figures 3.5 and B.19 constrain targets X and Y to be subsequences of zebrafish mRNA sequences *tpm3* (2175 nt) and *desma* (1798 nt), respectively. Zebrafish mRNA sequences were obtained from the National Center for Biotechnology Information (NCBI) supercitemcentyre02. Sequences are listed below 5′ to 3′. Table B.3 lists the sequence domains constrained by window constraints for each reaction pathway, the number of candidate windows within each mRNA, and the number of these candidate windows that also satisfy the composition constraints, the pattern prevention constraints, or both the composition and pattern prevention constraints.

mRNA: **tropomyosin 3** (*tpm3*)

Sequence: gAACACUAUUAgCUAUUUgUAgUACUCUAAAgAggACUgCAgAACgCAUCgCAgUAgUggUgAAAAgCCgUgCgUgCgCgUgAAACAUCUg
AUCCUCACgUUACUUCCACUCgCUCUgCgUUUgACUUgUUggCggggCgUUggUgCCUUggACUUUUUUUCCUCCUUCUCUUCUUCgCggCUCggUCCACUA
CgCUgCUCgAgAggAAUCUgCUUUAUUCgACCACACUACUCCUAAAgUAACACAUUAAAAUggCCggAUCAAACAgCAUCgAUgCAgUUAAgAgAAAAAUCAA
AgUUUUACAACAgCAAgCAgAUgAggCAgAAgAAAgAgCCgAgAUUUUgCAgAgACAggUCgAggAggAgAAgCgUgCCAgggAgCAggCUgAggCAgAggUg
gCUUCUCUgAACAggCgUAUCCAgCUggUUgAggAggAgUUggAUCgUgCUCAggAgAgACUggCCACAgCCCUgCAAAAgCUggAggAAgCCgAgAAggCCg
CAgAUgAgAgCgAgAgAgggAUgAAggUgAUUgAgAACAgggCUCUgAAggAUgAggAgAAgAUggAgCUgCAggAgAUCCAgCUUAAggAggCCAAgCACAU
UgCUgAggAggCUgACCgCAAAUAUgAAgAggUggCUCgUAAgCUggUgAUCgUUgAgggAgAgUUggAgCgUACAgAggAgAgAgCAgAgCUUgCAgAgAgC
CAUgUCAAgCAgAUggAggAggAgCUgAgAgCUCUUgACCAgACACUgAAgACUCUUCAggCCUCAgAggAgAAgUAUUCCCAgAAggAggACAAgUAUgAgg
AAgAAAUCAAgAUCCUCACUgAUAAgCUgAAggAggCUgAgACCCgUgCAgAgUUUgCUgAgAggUCUgUggCCAAACUggAgAAAACCAUUgAUgAUUUggA
AgAgAAACUgAgAgAUgCUAAAgAggAgAACAUCAAgAUCCAUgCUACUUUggACCAgACCCUgAgCgAgCUCAAUAgUUUCUAAAgAAgACCUggAgCAgAA
AAAAggCCUUUUCUUCCCUUCUUgACUCCCUCAUCUCAUUUUggUUUCUUUgUCUCUgCACAUCUgAUUCUCCCCCUUUUUUUUCUUCUCUUCUUCUgCUgg
AggAUAAgCUCACCAAgCCAACCAgCAAAAAUgUggUgCCUCUCAAUUUUUCCAAACUACUAUUCCAAgUgAUUUgAgAAAUgAUCUACUACgAUACUCCUCA
AgAgUCAAAUgUUgACCUCggggAgCCUUUUUUggUAUUgCUCCAUgAUCAgAgCUUUACgAgCUAgUgUUUUUUCUgCAUAUCAgCCCAAACUCUCAAUgAU
AAUUUUACUggAggCUgAUUUUUgUAAAAUUUUgUgCCAUAAAAgCCUUgUUggCUUgUCUCUUgCUUggCUUUAgAUCAUUCUCAAgCCAUUUUUUUCCUgC
UgUUgCUCUgACACAggUUgUUUUUgCUggUCUUgUUggUgCCUgAUCCACUgCUAUCCUUUUCACACCUCUUUUUUUUUUUUCUUCAUCCUgCACAAgUUUC
UgCUgCCUgUUAgUCggCAUCACCggUUUUgggACCAAAACCACAUCAUgUggUCUgUAACAgUAUgCACAACCAUgCCgUgAggACCAAAUUUgUUUUAUUA
UUgUUAUUAUUAUUAAAAgCCUUUgCUUCCAUUCggAgUUUgUUUUUUUgAgUAAUAUAUgUAUUCAUUgUUUgggUCgAAUCCCCUUgCUUUUUUAACACAA
AUgUUUUgCAAACCACUAUUUgAAAUggUgCACUgUUAUgggCUUAUggUgAgCAgAUgAggCCAAgUCAUggUUUCUUCAUUAUAAUUUUCUUUUCAUUUgC
UUUAAAgAgCCAUAUUCUACCCAgggAAgAAAggUUgAAgUUgUUUUgUUUUUUUACCgUgAgUUCAAAgCAgUggCACUgCCAgAUUUAAAAggUUCAAAAg
CCgUgCAgAUCUAAAAUAUgUAUUAUgAACACAgUAAUgggAgCgAAUUgUAACACUUAAUAgUAUACAAAUUUAAgAAACAggggUgAACACAUAgUUUUAA
CUggAAAAAgCCCACAAUgAUgUgUAAUCACUUUgUUACUgUCUgUAUCUUgUgUAAUgAUACCUAAAUUCUUUUUUUAAAUAAAAACCAUgAUUUUUACUgU
CACUgAAAAAAAAAAAAAAAAAA

mRNA: **desmin a** (*desma*)

Sequence: CAUUUACACAgCgUACAAACCCAACAggCCCAgUCAUgAgCACgAAAUAUUCAgCCUCCgCCgAgUCggCgUCCUCUUACCgCCgCACCUU
UggCUCAggUUUgggCUCCUCUAUUUUCgCCggCCACggUUCCUCAggUUCCUCUggCUCCUCAAgACUgACCUCCAgAgUUUACgAggUgACCAAgAgCUCC
gCUUCUCCCCAUUUUUCCAgCCACCgUgCgUCCggCUCUUUCggAggUggCUCggUggUCCgUUCCUACgCUggCCUUggUgAgAAgCUggAUUUCAAUCUgg
CUgAUgCCAUAAACCAggACUUCCUCAACACgCgUACUAAUgAgAAggCCgAgCUCCAgCACCUCAAUgACCgCUUCgCCAgCUACAUCgAgAAggUgCgCUU
CCUCgAgCAgCAgAACUCUgCCCUgACggUggAgAUUgAgCgUCUgCggggUCgCgAgCCCACCCgUAUUgCAgAgCUgUACgAggAggAgAUgAgAgAgCUg
CgCggACAggUggAggCACUgACCAAUCAgAgAUCCCgUgUggAgAUCgAgAgggACAACCUAgUCgAUgACCUACAgAAACUAAAgCUCAgACUUCAAgAgg
AgAUCCACCAgAAAgAggAAgCUgAAAACAACCUUUCUgCUUUCAgAgCUgAUgUCgAUgCUgCCACUCUggCCAggCUggACCUggAAAgACgUAUCgAggg
UCUUCACgAAgAgAUUgCAUUCCUCAggAAgAUUCAUgAggAggAgAUCCgUgAgCUgCAgAACCAgAUgCAggAgAgUCAggUgCAgAUCCAAAUggACAUg
UCCAAACCAgACCUgACUgCggCCCUCAgAgACAUUCgCCUgCAgUACgAggCUAUCgCUgCCAAgAAUAUCAgCgAggCCgAggACUggUAUAAgUCUAAgg
UUUCAgAUUUgAACCAggCAgUgAACAAgAAUAACgAggCUCUCAgAgAAgCCAAgCAggAgACCAUgCAgUUCCgUCACCAgCUCCAgUCCUACACCUgCgA
gAUUgACUCUCUCAAgggCACCAAUgAgUCUCUgAggAggCAAAUgAgUgAgAUggAggAgCggCUgggACgUgAggCCggUggUUAUCAggACACUAUCgCC
CgUCUCgAggCUgAgAUCgCAAAAAUgAAAgACgAgAUggCCCgCCACCUCCgCgAgUACCAggAUCUgCUgAAUgUgAAgAUggCUCUggAUgUggAgAUCg
CCACCUACAggAAgCUUUUggAAggAgAggAgAgCAggAUCUCgCUgCCCgUgCAgUCCUUUUCAUCCCUgAgUUUCAgAgAgAgCAgUCCAgAgCAgCACCA
CCACCAgCAgCAgCAACCACAACgCUCAUCUgAAgUCCACUCCAAgAAAACAgUCCUgAUCAAgACCAUCgAgACCCgCgAUggCgAggUCgUCAgCgAgUCC
ACACAgCACCAgCAggACgUCAUgUAAAgCUUgAgAAACAgAUCgAgUUUCACAgAAUgCCUUgCAUUUUCACUgAUggCCUCAggCUUUUUUAAgCACACAC
CCAgUAUUgCCgUgACCCAUUACCgCAUgUggAUgACgCAUggAgACAAAAggAAAgUgAgCUgAAAAACCAgAgggAggAAAAgUggAAUggUgUgAUgCUg
AgCgUUCAgAAAgUggCCAgAUgAgCUCAgAgUUUCUgAUUUAAUgAAUgUAUgUgUgCgUgUgUgUgUggUUgggUCAUAUCUgAgACACUgUUCCACAgCA
ACAAAAACAAUAAAAUUCACUgUAUUUUCUCCUAAAAAAAAAAAAAAAAAAAAAAAAAA

Reaction pathway	Constrained sequence domains	Target function	Window length (nt)	Source mRNA	Candidate windows	Satisfying composition constraints	Satisfying pattern constraints	Satisfying composition and pattern constraints
Conditional self-assembly via hybridization chain reaction (HCR)	b*-a*	input X	36	*tpm3*	2140	565 (26%)	116 (5.4%)	89 (4.2%)
Boolean logic AND using toehold sequestration gates	a-b	input X	30	*tpm3*	2146	641 (30%)	190 (8.9%)	114 (5.3%)
	c-d	input Y	30	*desma*	1769	695 (39%)	377 (21%)	151 (8.5%)
Self-assembly of a 3-arm junction via catalytic hairpin assembly (CHA)	y*-b*-x*-a*	input X	36	*tpm3*	2140	565 (26%)	116 (5.4%)	89 (4.2%)
Boolean logic AND using a cooperative hybridization gate	a-b	input X	25	*tpm3*	2151	510 (24%)	298 (14%)	115 (5.4%)
	c-d	input Y	25	*desma*	1774	526 (30%)	501 (28%)	154 (8.7%)
Conditional Dicer substrate formation via shape and sequence transduction	a-b-c	input X	18	*tpm3*	2158	317 (15%)	576 (27%)	117 (5.4%)
	w-x-y-z	output Y	21	*desma*	1778	586 (33%)	634 (36%)	219 (12%)

Table B.3: Window constraints for each reaction pathway based on mRNAs X and Y, as well as number of candidate windows satisfying: composition constraints ($\mathtt{S} \in [0.45, 0.55]$), pattern constraints (prevent {AAAA, CCCC, GGGG, UUUU, KKKKKK, MMMMMM, RRRRRR, SSSSSS, WWWWWW, YYYYYY}), or both.

B.2.6 Robustness of Predictions to Model Perturbations

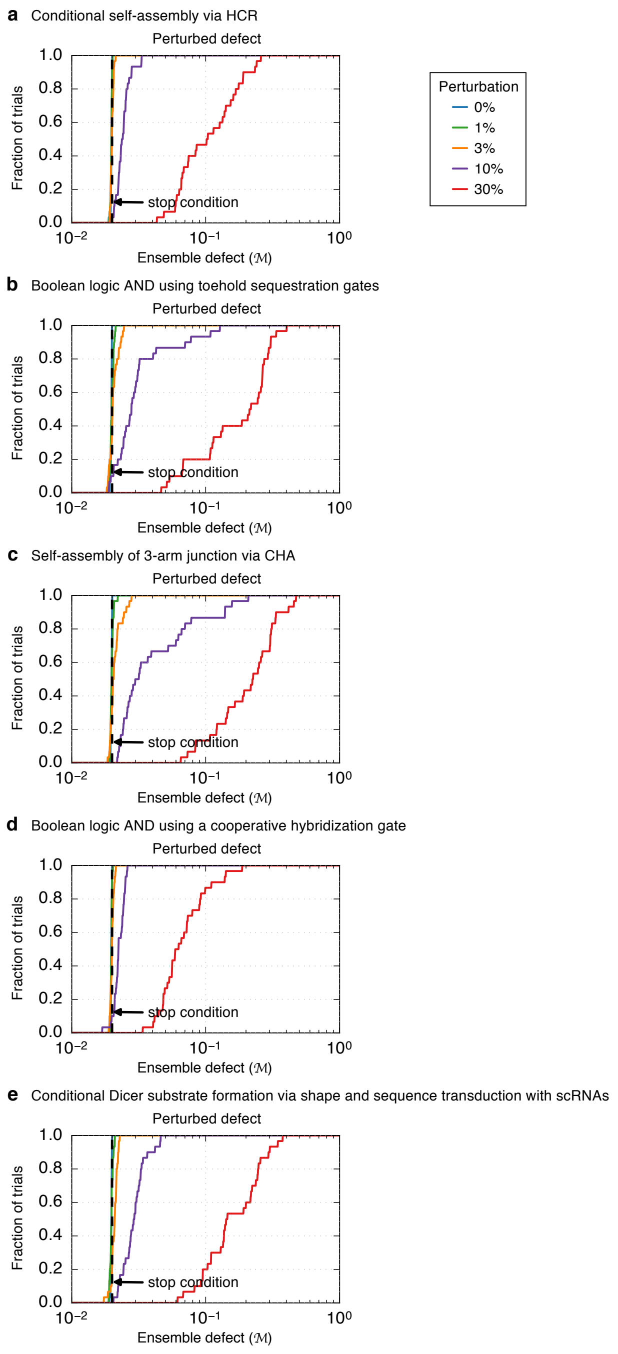


Figure B.20: Robustness of design quality predictions to perturbations in model parameters ($N = 8$ orthogonal systems). For each design trial, the median multistate test tube ensemble defect was calculated over 100 perturbed physical models (each parameter perturbed by Gaussian noise with a standard deviation of 0, 1, 3, 10, or 30% of the parameter modulus). The stop condition is depicted as a dashed black line. (a) Conditional self-assembly via HCR. (b) Boolean logic AND using toehold sequestration gates. (c) Self-assembly of 3-arm junction via CHA. (d) Boolean logic AND using a cooperative hybridization gate. (e) Conditional Dicer substrate formation via shape and sequence transduction with scRNAs.

B.3 Additional Design Studies

Here, we compare the performance of the current constrained multistate test tube design algorithm to that of the previously published test tube design algorithm[1] on the subsidiary design problems of test tube design and complex design (see Table 3.3 for a comparison of the design ensembles).

B.3.1 Performance for Test Tube Design

Test tube design is a special case of multistate test tube design in which the design ensemble contains only one target test tube containing an arbitrary number of on-target and off-target complexes (Table 3.3). For test tube design comparisons, we use the *engineered test set* and *random test set* of target test tubes provided with Reference 1. For the engineered test set, each on-target structure was randomly generated with stem and loop sizes randomly selected from a distribution of sizes representative of the nucleic acid engineering literature. For the random test set, each on-target structure was generated by calculating the minimum free energy structure of a different random RNA sequence in 1 M Na^+ at 37 °C. Within each target test tube, there are two on-target dimers (each with a target concentration of 1 μM) and a total of 106 off-target monomers, dimers, trimers and tetramers (each with vanishing target concentration), representing all complexes of up to $L_{\max} = 4$ strands (excluding the two on-target dimers). For each test set, 50 target test tubes were generated for each on-target dimer size, $\{50, 100, 200, 400\}$ nt, with all strands the same length in each target test tube. The structural properties of the on-target structures in the engineered and random test sets are summarized in Supplementary Section S3 of Reference 1. Five design trials were run for each test case using stop condition $f_{\text{stop}} = 0.01$ (i.e., no more than 1% of nucleotides incorrectly paired at equilibrium). For test tube design, Figures B.21 and B.22 demonstrate that the performance of the current algorithm and the previously published test tube design algorithm[1] is similar on the engineered and random test sets.

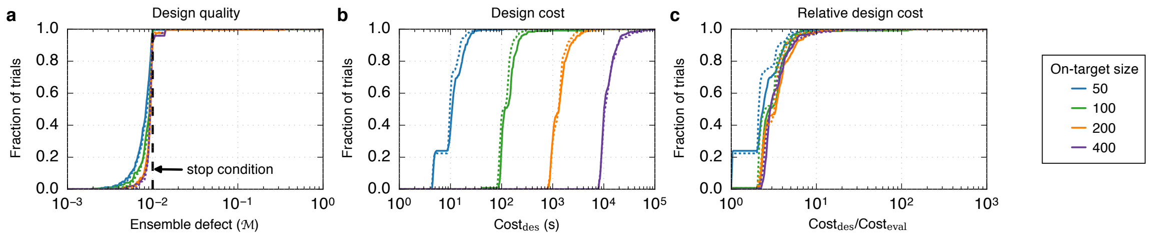


Figure B.21: Algorithm performance for test tube design on the engineered test set. Comparison of the current multistate test tube design algorithm (solid lines) to the previously published test tube design algorithm[1] (dotted lines). (a) Design quality. The stop condition is depicted as a dashed black line. (b) Design cost. (c) Cost of sequence design relative to a single evaluation of the objective function. Design conditions: RNA in 1 M Na$^+$ at 37 °C.

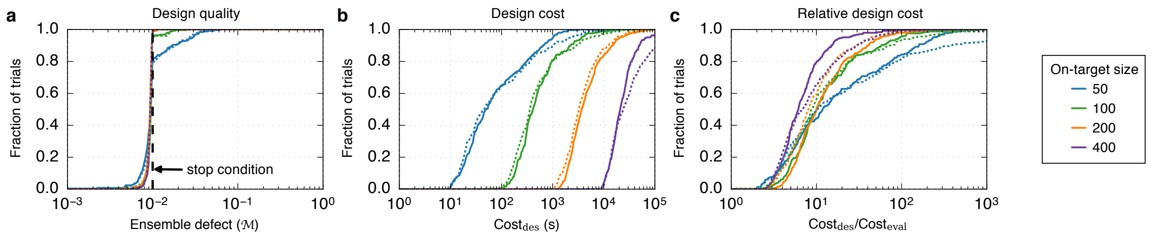


Figure B.22: Algorithm performance for test tube design on the random test set. Comparison of the current multistate test tube design algorithm (solid lines) to the previously published test tube design algorithm[1] (dotted lines). (a) Design quality. The stop condition is depicted as a dashed black line. (b) Design cost. (c) Cost of sequence design relative to a single evaluation of the objective function. Design conditions: RNA in 1 M Na$^+$ at 37 °C.

B.3.2 Performance for Complex Design

Complex design is a special case of multistate test tube design in which the design ensemble contains one target test tube containing one on-target complex and no off-target complexes (Table 3.3). For complex design comparisons, we use the (dimer) on-target structures from the target test tubes in the engineered and random test sets described above. Hence, for the studies, each target test tube contains a single on-target dimer and no off-targets; there are 50 on-target structures of each size $\{50, 100, 200, 400\}$ nt (using the first-listed on-target dimer in each of the 50 target test tubes of Section B.3.1). Five design trials were run for each test case using stop condition $f_{\text{stop}} = 0.01$ (i.e., no more than 1% of nucleotides incorrectly paired at equilibrium). For complex design, Figures B.23 and B.24 demonstrate that the performance of the current algorithm and the previously published test tube design algorithm[1] is similar for the on-target structures in the engineered and random test sets.

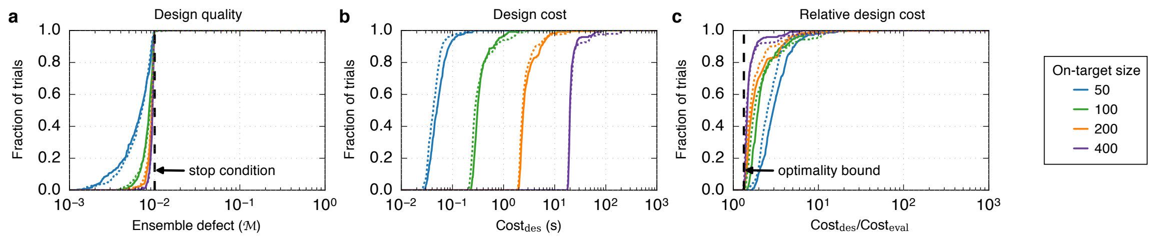


Figure B.23: Algorithm performance for complex design using on-target structures from the engineered test set. Comparison of the current multistate test tube design algorithm (solid lines) to the previously published test tube design algorithm[1] (dotted lines). (a) Design quality. The stop condition is depicted as a dashed black line. (b) Design cost. (c) Cost of sequence design relative to a single evaluation of the objective function. The optimality bound[4] is depicted as a dashed black line. Design conditions: RNA in 1 M Na$^+$ at 37 °C.

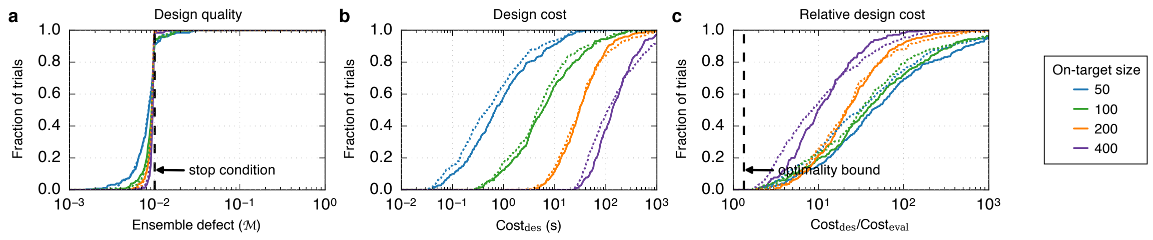


Figure B.24: Algorithm performance for complex design using on-target structures from the random test set. Comparison of the current multistate test tube design algorithm (solid lines) to the previously published test tube design algorithm[1] (dotted lines). (a) Design quality. The stop condition is depicted as a dashed line. (b) Design cost. (c) Cost of sequence design relative to a single evaluation of the objective function. The optimality bound[4] is depicted as a dashed black line. Design conditions: RNA in 1 M Na$^+$ at 37 °C.

Bibliography

[1] B. R. Wolfe and N. A. Pierce. “Nucleic acid sequence design for a test tube of interacting nucleic acid strands”. In: *ACS Synth. Biol.* 4.10 (2015), pp. 1086–1100.

[2] R. M. Dirks, J. S. Bois, J. M. Schaeffer, E. Winfree, and N. A. Pierce. “Thermodynamic Analysis of Interacting Nucleic Acid Strands”. In: *SIAM Rev.* 49.1 (2007), pp. 65–88.

[3] J. N. Zadeh, C. D. Steenberg, J. S. Bois, B. R. Wolfe, M. B. Pierce, A. R. Khan, R. M. Dirks, and N. A. Pierce. “NUPACK: Analysis and Design of Nucleic Acid Systems”. In: *J. Comput. Chem.* 32.1 (2011), pp. 170–173.

[4] J. N. Zadeh, B. R. Wolfe, and N. A. Pierce. “Nucleic Acid Sequence Design via Efficient Ensemble Defect Optimization”. In: *J. Comput. Chem.* 32 (2011), pp. 439–452.

[5] M. J. Serra and D. H. Turner. “Predicting thermodynamic properties of RNA”. In: *Methods Enzymol.* 259 (1995), pp. 242–261.

[6] D. H. Mathews, J. Sabina, M. Zuker, and D. H. Turner. “Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure”. In: *J. Mol. Biol.* 288 (1999), pp. 911–940.

[7] M. Zuker. “Mfold web server for nucleic acid folding and hybridization prediction”. In: *Nucleic Acids Res.* 31.13 (2003), pp. 3406–3415.

[8] J. SantaLucia Jr. and D. Hicks. “The thermodynamics of DNA structural motifs”. In: *Annu. Rev. Biophys. Biomol. Struct.* 33 (2004), pp. 415–440.

[9] R. T. Koehler and N. Peyret. “Thermodynamic Properties of DNA Sequences: Characteristic Values for the Human Genome”. In: *Bioinformatics* 21.16 (2005), pp. 3333–3339.

[10] A. J. Genot, D. Y. Zhang, J. Bath, and A. J. Turberfield. “Remote Toehold: A Mechanism for Flexible Control of DNA Hybridization Kinetics”. In: *J. Am. Chem. Soc.* 133.7 (2011), pp. 2177–2182.

[11] A. J. Genot, J. Bath, and A. J. Turberfield. “Reversible Logic Circuits Made of DNA”. In: *J. Am. Chem. Soc.* 133.50 (2011), pp. 20080–20083.

[12] C. J. Delebecque, P. A. Silver, and A. B. Lindner. “Designing and using RNA scaffolds to assemble proteins in vivo”. In: *Nat. Protoc.* 7.10 (2012), pp. 1797–1807.

[13] D. G. Greene, J. W. Keum, and H. Bermudez. “The Role of Defects on the Assembly and Stability of DNA Nanostructures”. In: *Small* 8.9 (2012), pp. 1320–1325.

[14] A. Padirac, T. Fujii, and Y. Rondelez. “Quencher-free multiplexed monitoring of DNA reaction circuits”. In: *Nucleic Acids Res.* 40.15 (2012).

[15] H. Tang, R. Deschner, P. Allen, Y. J. Cho, P. Sermas, A. Maurer, A. D. Ellington, and C. G. Willson. “Analysis of DNA-Guided Self-Assembly of Microspheres Using Imaging Flow Cytometry”. In: *J. Am. Chem. Soc.* 134.37 (2012), pp. 15245–15248.

[16] X. J. Zhang and V. K. Yadavalli. “Functional self-assembled DNA nanostructures for molecular recognition”. In: *Nanoscale* 4.7 (2012), pp. 2439–2446.

[17] D. B. Goodman, G. M. Church, and S. Kosuri. “Causes and Effects of N-Terminal Codon Bias in Bacterial Genes”. In: *Science* 342.6157 (2013), pp. 475–479.

[18] X. W. Xu and X. R. Yang. “Reversion of DNA strand displacement using functional nucleic acids as toeholds”. In: *Chem. Commun.* 50.7 (2014), pp. 805–807.

[19] R. M. Dirks and N. A. Pierce. “Triggered Amplification by Hybridization Chain Reaction”. In: *Proc. Natl. Acad. Sci. USA* 101.43 (2004), pp. 15275–15278.

[20] V. Patzel, S. Rutz, I. Dietrich, C. Köberle, A. Sheffold, and S. Kaufmann. “Design of siRNAs producing unstructured guide-RNAs results in improved RNA interference efficiency”. In: *Nat. Biotechnol.* 23.11 (2005), pp. 1440–1444.

[21] R. Penchovsky and R. Breaker. “Computational design and experimental validation of oligonucleotide-sensing allosteric ribozymes”. In: *Nat. Biotechnol.* 23.11 (2005), pp. 1424–1433.

[22] S. Venkataraman, R. M. Dirks, P. W. K. Rothemund, E. Winfree, and N. A. Pierce. “An autonomous polymerization motor powered by DNA hybridization”. In: *Nat. Nanotechnol.* 2 (2007), pp. 490–494.

[23] P. Yin, H. M. T. Choi, C. R. Calvert, and N. A. Pierce. “Programming biomolecular self-assembly pathways”. In: *Nature* 451.7176 (2008), pp. 318–322.

[24] H. M. Salis, E. A. Mirsky, and C. A. Voigt. “Automated design of synthetic ribosome binding sites to control protein expression”. In: *Nat. Biotechnol.* 27.10 (2009), pp. 946–950.

[25] H. M. T. Choi, J. Y. Chang, L. A. Trinh, J. E. Padilla, S. E. Fraser, and N. A. Pierce. “Programmable in situ amplification for multiplexed imaging of mRNA expression”. In: *Nat. Biotechnol.* 28.11 (2010), pp. 1208–12.

[26] B. L. Li, A. D. Ellington, and X. Chen. “Rational, modular adaptation of enzyme-free DNA circuits to multiple detection methods”. In: *Nucleic Acids Res.* 39.16 (2011), e110.

[27] J. Choi, K. R. Love, Y. Gong, T. M. Gierahn, and J. C. Love. “Immuno-Hybridization Chain Reaction for Enhancing Detection of Individual Cytokine-Secreting Human Peripheral Mononuclear Cells”. In: *Anal. Chem.* 83.17 (2011), pp. 6890–6895.

[28] J. Dong, X. Cui, Y. Deng, and Z. Tang. “Amplified Detection of Nucleic Acid by G-Quadruplex Based Hybridization Chain Reaction”. In: *Biosens. Bioelectron.* 38.1 (2012), pp. 258–263.

[29] T. Nishimura, Y. Ogura, and J. Tanida. “Fluorescence resonance energy transfer-based molecular logic circuit using a DNA scaffold”. In: *Appl. Phys. Lett.* 101 (2012), p. 233703.

[30] M. Schade, A. Knoll, A. Vogel, O. Seitz, J. Liebscher, D. Huster, A. Herrmann, and A. Arbuzova. “Remote Control of Lipophilic Nucleic Acids Domain Partitioning by DNA Hybridization and Enzymatic Cleavage”. In: *J. Am. Chem. Soc.* 134.50 (2012), pp. 20490–20497.

[31] J. R. Vieregg, H. M. Nelson, B. M. Stoltz, and N. A. Pierce. “Selective nucleic acid capture with shielded covalent probes”. In: *J. Am. Chem. Soc.* 135.26 (2013), pp. 9691–9699.

[32] L. M. Hochrein, M. Schwarzkopf, M. Shahgholi, P. Yin, and N. A. Pierce. “Conditional Dicer substrate formation via shape and sequence transduction with small conditional RNAs”. In: *J. Am. Chem. Soc.* 135.46 (2013), pp. 17322–17330.

[33] A. J. Genot, J. Bath, and A. J. Turberfield. “Combinatorial Displacement of DNA Strands: Application to Matrix Multiplication and Weighted Sums”. In: *Angew. Chem., Int. Ed.* 52.4 (2013), pp. 1189–1192.

[34] G. D. Hamblin, A. A. Hariri, K. M. M. Carneiro, K. L. Lau, G. Cosa, and H. F. Sleiman. "Simple Design for DNA Nanotubes from a Minimal Set of Unmodified Strands: Rapid, Room-Temperature Assembly and Readily Tunable Structure". In: *ACS Nano* 7.4 (2013), pp. 3022–3028.

[35] C. C. Santini, J. Bath, A. M. Tyrrell, and A. J. Turberfield. "A clocked finite state machine built from DNA". In: *Chem. Commun.* 49.3 (2013), pp. 237–239.

[36] H. M. T. Choi, V. A. Beck, and N. A. Pierce. "Next-generation in situ hybridization chain reaction: higher gain, lower cost, greater durability". In: *ACS Nano* 8.5 (2014), pp. 4284–4294.

[37] Y. S. Jiang, S. Bhadra, B. L. Li, and A. D. Ellington. "Mismatches Improve the Performance of Strand-Displacement Nucleic Acid Circuits". In: *Angew. Chem., Int. Ed.* 53.7 (2014), pp. 1845–1848.

[38] C. Geary, P. W. K. Rothemund, and E. S. Andersen. "A single-stranded architecture for cotranscriptional folding of RNA nanostructures". In: *Science* 345.6198 (2014), pp. 799–804.

[39] A. A. Green, P. A. Silver, J. J. Collins, and P. Yin. "Toehold Switches: De-Novo-Designed Regulators of Gene Expression". In: *Cell* 159.4 (2014), pp. 925–939.

[40] J. M. Hu, Y. J. Yu, J. C. Brooks, L. A. Godwin, S. Somasundaram, F. Torabinejad, J. Kim, C. Shannon, and C. J. Easley. "A Reusable Electrochemical Proximity Assay for Highly Selective, Real-Time Protein Quantitation in Biological Matrices". In: *J. Am. Chem. Soc.* 136.23 (2014), pp. 8467–8474.

[41] R. R. F. Machinek, T. E. Ouldridge, N. E. C. Haley, J. Bath, and A. J. Turberfield. "Programmable energy landscapes for kinetic control of DNA strand displacement". In: *Nat Commun* 5 (2014), p. 5324.

[42] E. Franco, G. Giordano, P. O. Forsberg, and R. M. Murray. "Negative Autoregulation Matches Production and Demand in Synthetic Transcriptional Networks". In: *ACS Synth. Biol.* 3.8 (2014), pp. 589–599.

[43] B. Koos, G. Cane, K. Grannas, L. Lof, L. Arngarden, J. Heldin, C. M. Clausson, A. Klaesson, M. K. Hirvonen, F. M. S. de Oliveira, V. O. Talibov, N. T. Pham, M. Auer, U. H. Danielson, J. Haybaeck, M. Kamali-Moghaddam, and O. Soderberg. "Proximity-dependent initiation of hybridization chain reaction". In: *Nat Commun* 6 (2015), p. 7294.

[44] J. G. Zalatan, M. E. Lee, R. Almeida, L. A. Gilbert, E. H. Whitehead, M. La Russa, J. C. Tsai, J. S. Weissman, J. E. Dueber, L. S. Qi, and W. A. Lim. "Engineering Complex Synthetic Transcriptional Programs with CRISPR RNA Scaffolds". In: *Cell* 160.1-2 (2015), pp. 339–350.

[45] R. P. Galimidi, J. S. Klein, M. S. Politzer, S. Y. Bai, M. S. Seaman, M. C. Nussenzweig, A. P. West, and P. J. Bjorkman. "Intra-Spike Crosslinking Overcomes Antibody Evasion by HIV-1". In: *Cell* 160.3 (2015), pp. 433–446.

[46] T. Raschle, C. X. Lin, R. Jungmann, W. M. Shih, and G. Wagner. "Controlled Co-reconstitution of Multiple Membrane Proteins in Lipid Bilayer Nanodiscs Using DNA as a Scaffold". In: *ACS Chem. Biol.* 10.11 (2015), pp. 2448–2454.

[47] M. K. Takahashi, K. E. Watters, P. M. Gasper, T. R. Abbott, P. D. Carlson, A. A. Chen, and J. B. Lucks. "Using in-cell SHAPE-Seq and simulations to probe structure-function design principles of RNA transcriptional regulators". In: *RNA* 22.6 (2016), pp. 920–933.

[48] Y. J. Lee, A. Hoynes-O'Connor, M. C. Leong, and T. S. Moon. "Programmable control of bacterial gene expression with the combined CRISPR and antisense RNA system". In: *Nucleic Acids Res.* 44.5 (2016), pp. 2462–2473.

[49] M. Zuker and P. Stiegler. "Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information". In: *Nucleic Acids Res.* 9.1 (1981), pp. 133–147.

[50] J. McCaskill. "The equilibrium partition function and base pair binding probabilities for RNA secondary structure". In: *Biopolymers* 29 (1990), pp. 1105–1119.

[51] I. Hofacker, W. Fontana, P. Stadler, L. Bonhoeffer, M. Tacker, and P. Schuster. "Fast folding and comparison of RNA secondary structures". In: *Chem. Mon.* 125 (1994), pp. 167–188.

[52] R. B. Lyngso, M. Zuker, and C. N. S. Pedersen. "Fast evaluation of internal loops in RNA secondary structure prediction". In: *Bioinformatics* 15.6 (1999), pp. 440–445.

[53] R. M. Dirks and N. A. Pierce. "An algorithm for computing nucleic acid base-pairing probabilities including pseudoknots". In: *J. Comput. Chem.* 25 (2004), pp. 1295–1304.

[54] R. Dimitrov and M. Zuker. "Prediction of hybridization and melting for double-stranded nucleic acids". In: *Biophys. J.* 87.1 (2004), pp. 215–226.

[55] M. Andronescu, Z. Zhang, and A. Condon. "Secondary structure prediction of interacting RNA molecules". In: *J. Mol. Biol.* 345 (2005), pp. 987–1001.

[56] S. Bernhart, H. Tafer, U. Muckstein, C. Flamm, P. Stadler, and I. Hofacker. "Partition function and base pairing probabilities of RNA heterodimers". In: *Algorithm Mol. Biol.* 1.3 (2006).

[57] R. Dechter. *Constraint Processing*. New York: Morgan Kaufmann, 2003.

[58] G. Seelig, D. Soloveichik, D. Zhang, and E. Winfree. "Enzyme-free nucleic acid logic circuits". In: *Science* 314.5805 (2006), pp. 1585–1588.

[59] D. Y. Zhang. "Cooperative Hybridization of Oligonucleotides". In: *J. Am. Chem. Soc.* 133.4 (2011), pp. 1077–1086.

Appendix C

Design Visualization Utilities

This appendix describes two visualization utilies built using the Python Bokeh library[1] and one built using the Python NetworkX library[2].

C.1 Realtime Design Trajectory Visualization

When using the multiobjective design algorithm of Chapter 4, a user may set a filepath to act as a log file. Information is then logged after the following set of algorithm states is reached:

- A mutation is accepted.
- A mutation is rejected.
- Leaf sequences are reseeded.
- Sequence merging is successful.
- Sequence merging is unsuccessful.
- The current successfully merged sequence is better than the previous best sequence at a given depth in the decomposition forest.
- Complexes in Ψ^{active} are redecomposed.
- The current sequence is the best encountered sequence when evaluated over the full ensemble.
- The current sequence is worse than a previously sequence when evaluated over the full ensemble.
- An off-target is transferred to Ψ^{active} during ensemble refocusing.

Information is logged about the identity of the state, the current position in the tree (depth), the sizes of the sets Ψ^{active} and Ψ^{passive}, the time since the start of design, and the current objective function (or estimate). Additionally the sequence is logged as a JSON string representing a map of each domain to its sequence. Results are displayed in realtime using Bokeh[1] and data is parsed from the log file using Pandas[3]. Each of the above states corresponds to a different colored series of connected dots on a plot of the objective function (or estimate) vs. time since the beginning of design. Hovering over any of the dots produces a floating tooltip detailing the remainder of the logged information, sans the sequences (which can be read from the log file more easily). An example is shown in Figure C.1.

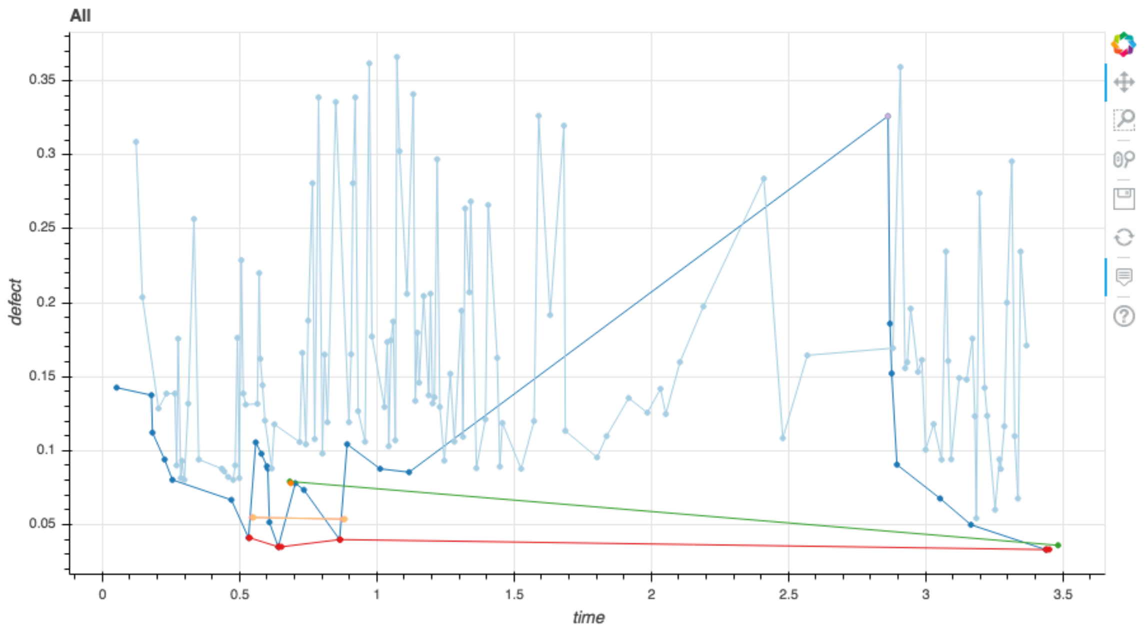


Figure C.1: Example trajectory for design of the conditional Dicer substrate formation via shape and sequence transduction reaction pathway. Each colored series corresponds to each time the design algorithm enters a particular state: dark blue for an accepted mutation, light blue for a rejected mutation, red for successful sequence merging, pink for unsuccessful sequence merging, light orange for redecomposition, dark orange for ensemble refocusing, and green for encountering the best sequence over the full ensemble.

We expect this to allow future developers of the design algorithm to visually inspect the effect of algorithm changes on the trajectory of benchmarking designs. Furthermore, it can act as as useful tool for users wishing to either examine the effects of changing constraints or tube specifications as well as allow preemption when a long running design is determined to be “good enough”.

C.2 Interactive Design Results Visualization

After a design job has finished, a vast amount of information about the final design state is returned as a Python object. This object can be explored directly in an interactive session or serialized to JSON and saved in a file. For designs with many complexes and test tubes, obtaining a gestalt view of the design by exploring the text in this file can be difficult (while still very useful for collecting the designed sequences). The JSON file can be read into our program for visualizing the results in an interactive and intuitive way. This program is built using Bokeh[1].

After entering the filename of their JSON output file, the user is presented with a test tube summary page containing a series of bar plots corresponding to the set of target test tubes Ω. The total height of each pair of stacked bars shows the normalized test tube defect $\mathcal{M}_h$ for each test tube h. This is partitioned into defect attributed to incorrect nucleotides in the ensemble of an on-target complex (structural defect) and nucleotides that have formed off-target complexes (concentration defect). This is shown in Figure C.2. A toggle switch allows redisplay of the data on a logarithmic y-axis, with bars replaced with circles, as shown in Figure C.3. Note that in the logarithmic case, the circles independently represent the structural and concentration defects, so they do not "stack". Hovering over any of the bars or circles reveals a tooltip with the numeric information and name of the tube.

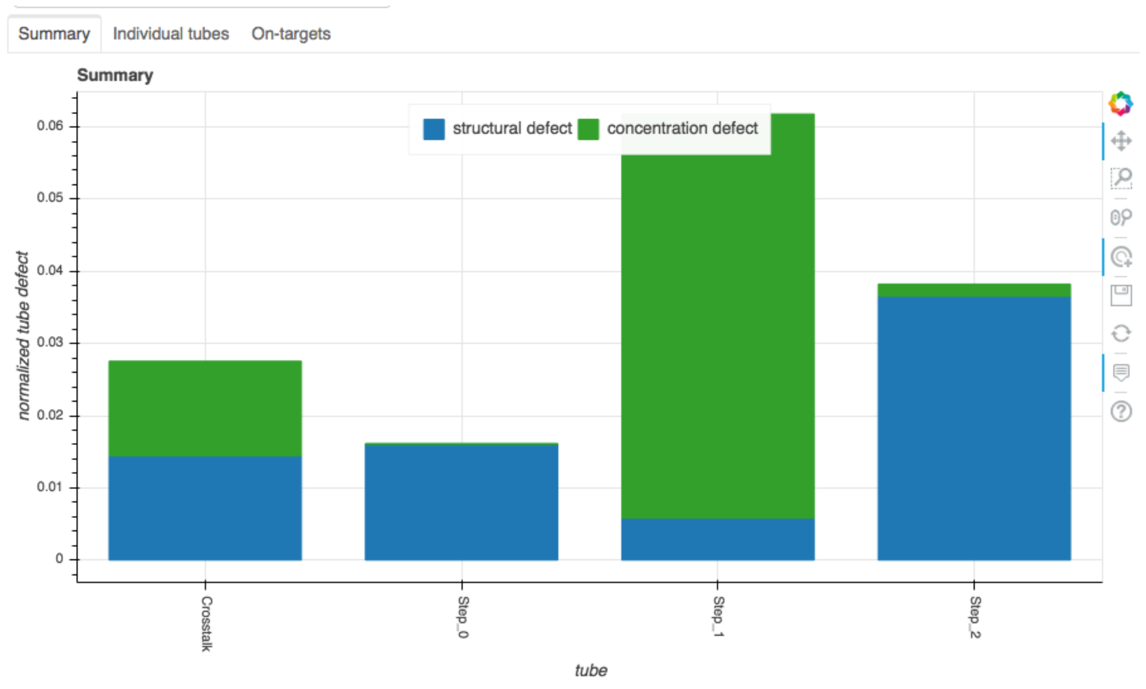


Figure C.2: The residuals of target test tubes on a linear scale. Structural defect shown in blue; concentration defect shown in green.

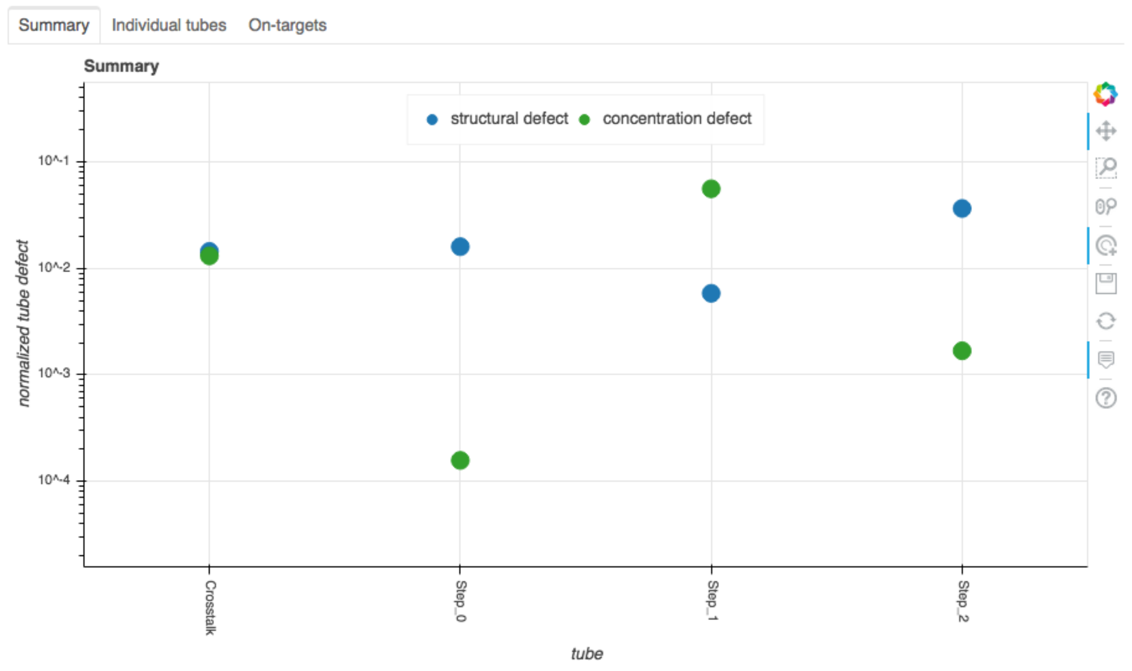
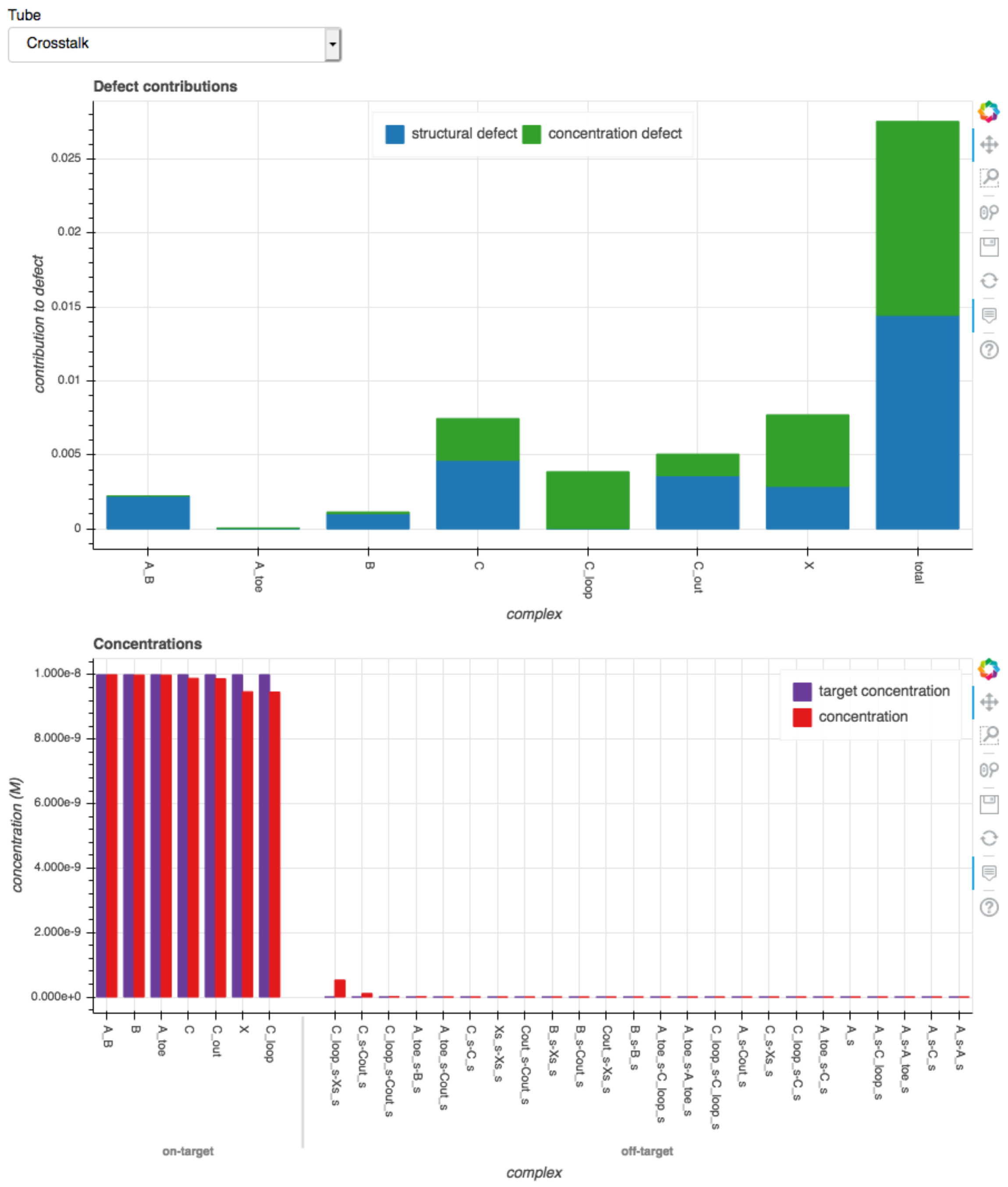


Figure C.3: The residuals of target test tubes on a linear scale. Structural defect shown in blue; concentration defect shown in green. Note: the hover tool shows numerical values for both structural and concentration defect components even if the circles occlude one another.

By clicking on either a bar or circle corresponding to a given test tube, the test tube summary is replaced with a test tube details page containing information about the complexes of the clicked test tube. The upper panel shows residual defects, divided into structural and concentration defect components for each of the on-targets in the test tube. The rightmost bar, labeled “total”, shows the sum of the structural and concentration components over all on-targets in the tube. The height of the stacked total bar is equivalent to the normalized test tube ensemble defect, $\mathcal{M}_h$, for this tube. The lower panel contains a second set of bar graphs depicting the target and actual concentrations of all on-targets and off-targets in the test tube. This is shown in Figure C.4. The results can be redisplayed on logarithmic scales as shown in Figure C.5. A user can threshold the results using the box zoom feature of the interactive concentration plot, which becomes important for test tubes with thousands of off-targets. This can be especially helpful for comparing the concentrations of off-targets in a test tube with very low concentration defect. Hovering over any of the bars, circles, or x’s reveals a tooltip with the numeric information and name of the complex. The results in a different test tube can be explored by selecting the other test tube’s name from a dropdown list, or by navigating back to the test tube summary page and clicking a different bar/circle.

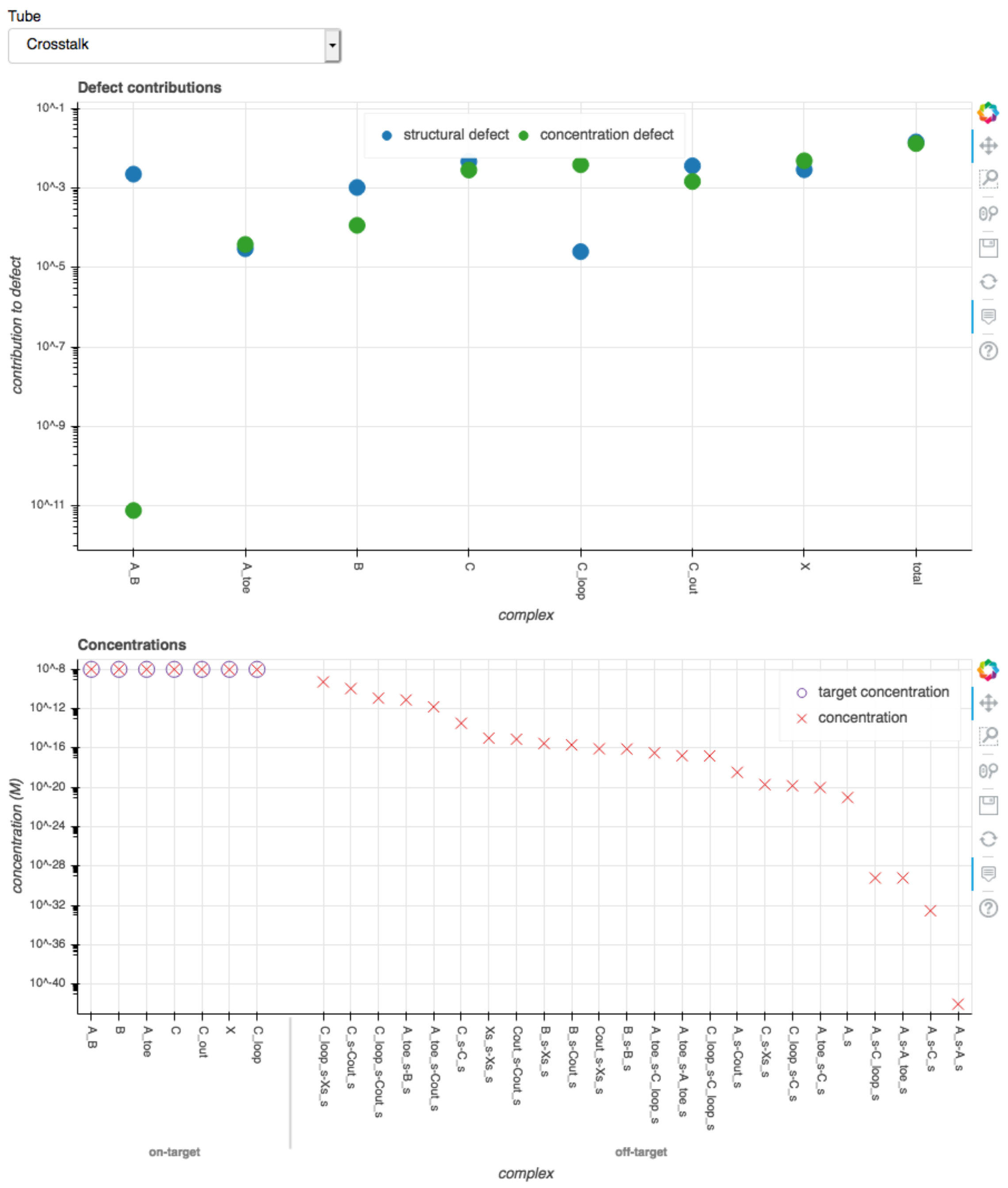
Finally, a user can compare the ensemble defects $n(\phi_j, s_j)$ of all on-targets in Ψ^{on} simultaneously by moving to the on-targets page. Both the ensemble defect in nucleotides and the normalized ensemble defect are shown to allow absolute and relative comparison. On-targets are sorted in both sets of bars in descending order of normalized complex ensemble defect. This is shown in Figure C.6. As before, the data can be simultaneously redisplayed on logarithmic axes, as shown in Figure C.7. Hovering over any of the bars or circles reveals a tooltip with the numeric information and name of the on-target complex.



Tube

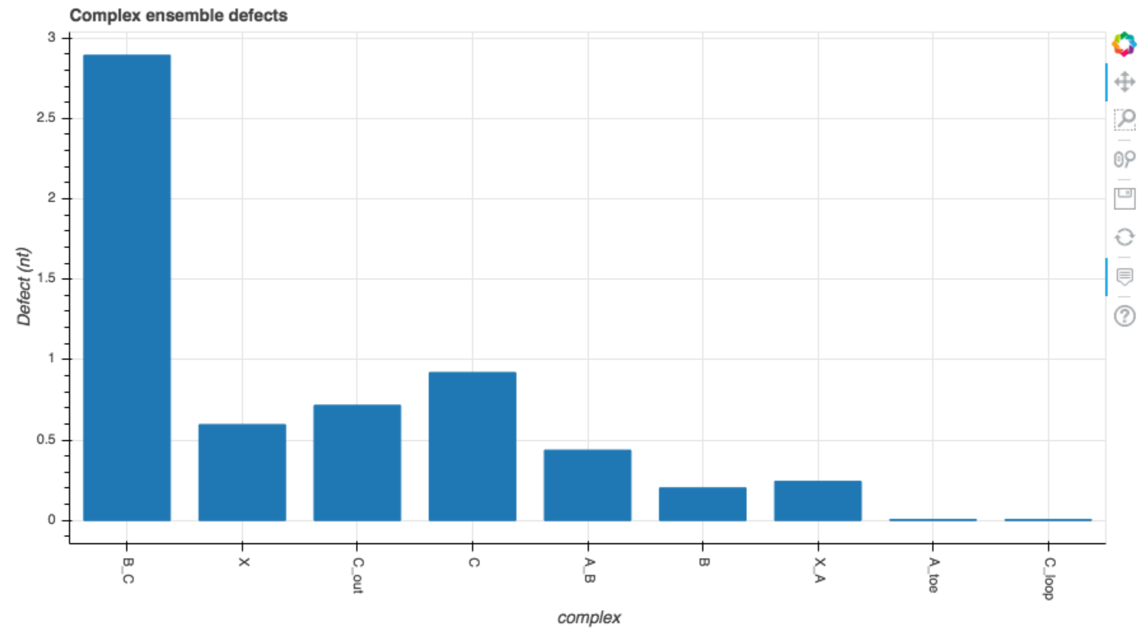
Crosstalk

Figure C.4: Detailed information about a target test tube on linear scales. Top: Structural defect (shown in blue) and concentration defect (shown in green) contributions of each on-target to the overall test tube ensemble defect. Bottom: Target concentration (shown in purple) and actual concentration (shown in red) of each complex in the test tube, partitioned into on-targets and off-targets and sorted within each subset in descending order of actual concentration.


Tube

Crosstalk

Figure C.5: Detailed information about a target test tube on logarithmic scales. Top: Structural defect (shown in blue) and concentration defect (shown in green) contributions of each on-target to the overall test tube ensemble defect. Note: the hover tool shows numerical values for both structural and concentration defect components even if the circles occlude one another. Bottom: Target concentration (shown as purple circles) and actual concentration (shown as red x's) of each complex in the test tube, partitioned into on-targets and off-targets and sorted within each subset in descending order of actual concentration.



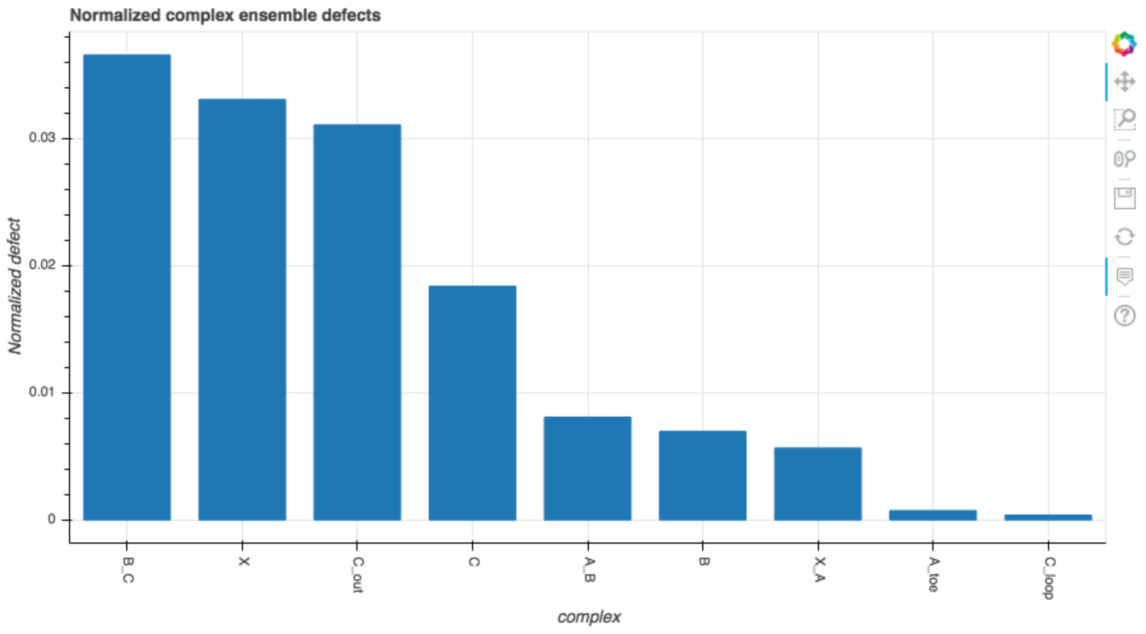
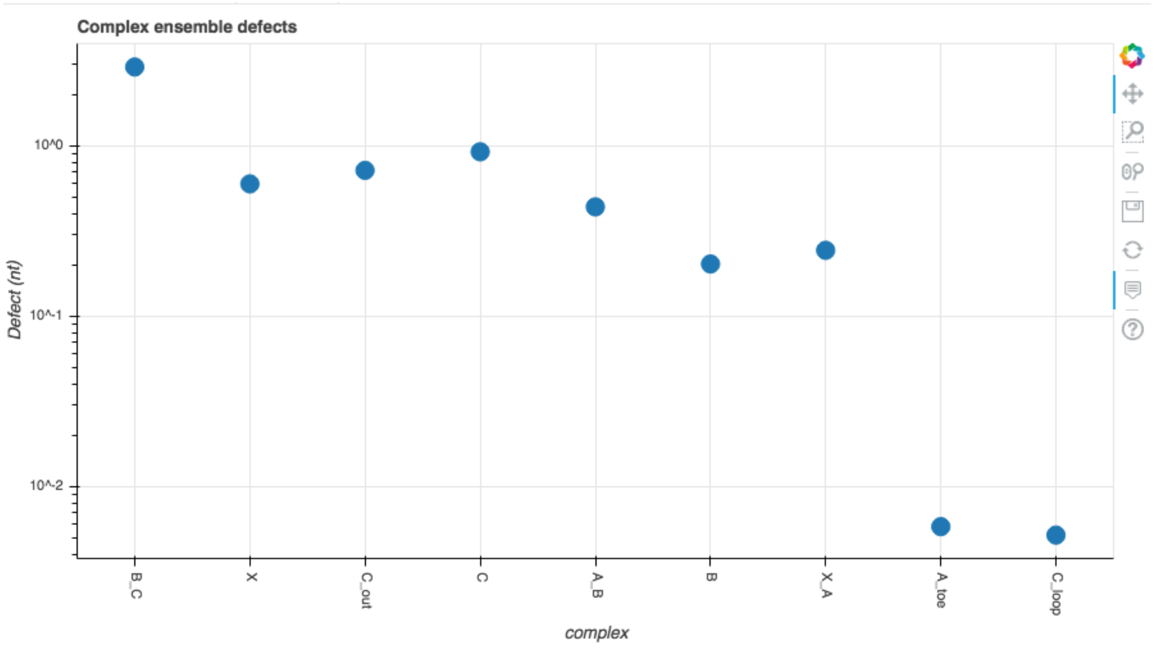


Figure C.6: Ensemble defects of all on-targets in the design on a linear scale. Top: Ensemble defect in nucleotides. Bottom: Ensemble defect normalized as a fraction of the nucleotides in the on-target.



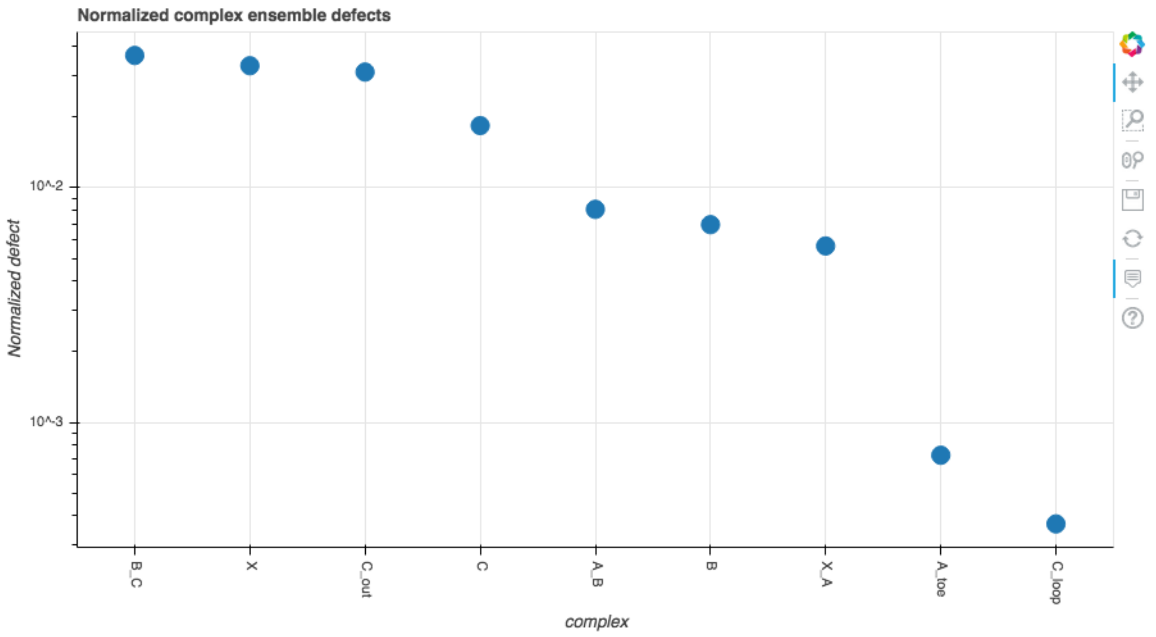


Figure C.7: Ensemble defects of all on-targets in the design on a logarithmic scale. Top: Ensemble defect in nucleotides. Bottom: Ensemble defect normalized as a fraction of the nucleotides in the on-target.

C.3 Hierarchical Decomposition Visualization

Addtionally, as both a debugging tool and exploratory analysis tool, we developed a software pipeline for visualizing the decomposition cost tree of a hierarchically decomposed complex. By setting an output file path, the multiobjective design algorithm emits textually serialized representations of the decomposition trees of complexes in Ψ^{active} during design. This is then read-in to Python with Pandas[3]. This data is then post-processed in Python to recreate the structure of the decomposition tree tagged with nodal information out of nested built-in Python types. This object can be explored with a depth-first search algorithm to compute the cost of the tree and then processed into a NetworkX[2] graph. NetworkX provides visualization functionality used to lay out the nodes in a force-directed fashion. Because it is a decomposition tree, it is possible to use a planar layout, but the options in NetworkX to do this lead to asymmetric representations of the tree, which is less desirable. Nodes are colored by the log of the number of nucleotides in the node and labeled with this number.

Examples of an efficient (Figure C.8) and inefficient (Figure C.9) decomposition are shown. We expect this will be a useful utility for developers to ensure that changes to the decomposition algorithm have not introduced bugs. Also, it should be useful for users to discover if the interplay between base pairing and sequence constraints is leading to expensive decompositions, which could either indicate a logical error in a design script or a particularly difficult design problem (e.g. a complex containing repeated complementary domains targeted to form one of multiple alternative domain-level structures).

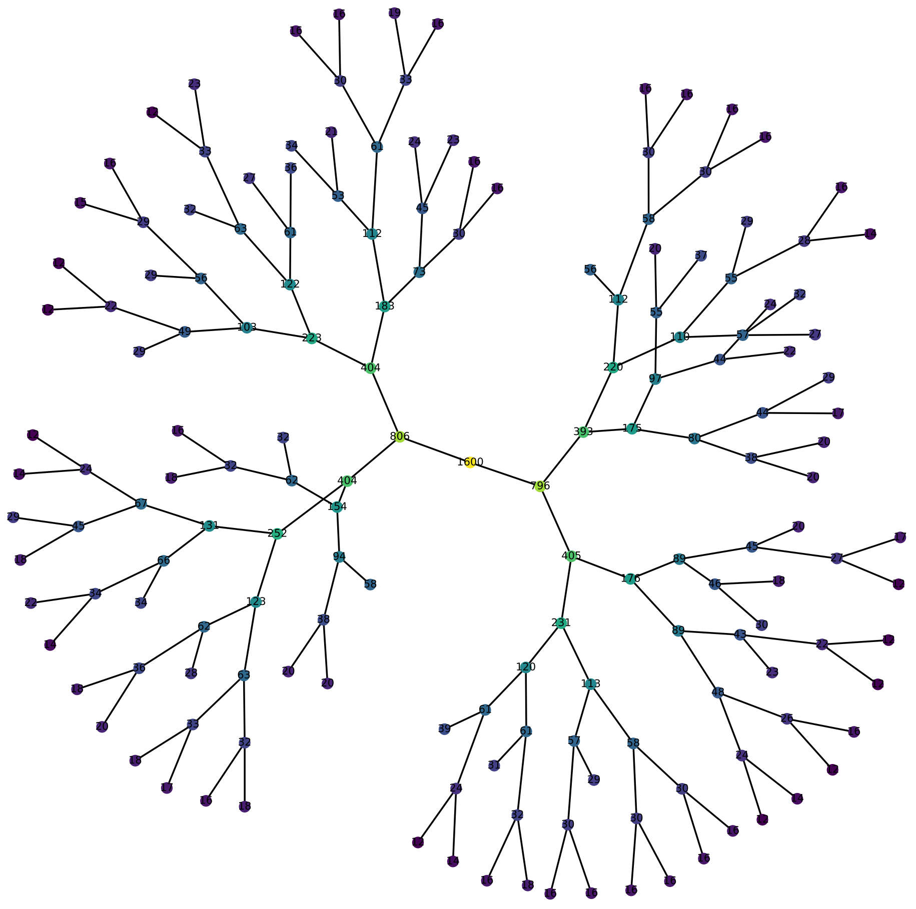


Figure C.8: Example of an efficient decomposition of a 1600 nt structure. The cost to compute the entire tree is 1.34× the cost to compute the root node.

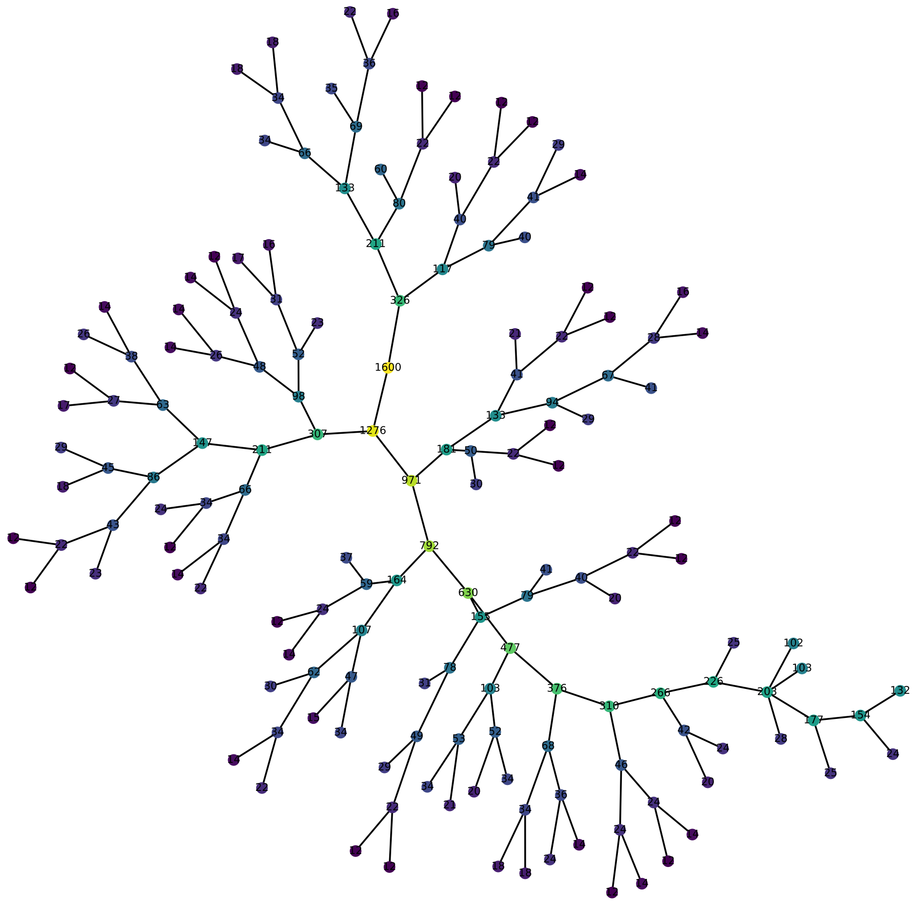


Figure C.9: Example of an inefficient decomposition of a 1600 nt structure. The cost to compute the entire tree is 2.00× the cost to compute the root node.

Bibliography

[1] Bokeh Development Team. *Bokeh: Python library for interactive visualization*. 2018. URL: https://bokeh.pydata.org/en/latest/.

[2] A. Hagberg, P. Swart, and D. S Chult. *Exploring network structure, dynamics, and function using NetworkX*. Tech. rep. Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[3] W. McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference*. Ed. by S. van der Walt and J. Millman. 2010, pp. 51–56.