4-25-2016

On the Systematic Design and Analysis of Artificial Molecular Machines

Pouya Tavousi
University of Connecticut - Storrs, pouya.tavousi@gmail.com

On the Systematic Design and Analysis of Artificial Molecular Machines

Pouya Tavousi, Ph.D.

University of Connecticut, 2016

ABSTRACT

Inside every cell, ribosomes, the natural molecular factories, use genetic data as "building instructions" to assemble 20 types of amino acids into long chain molecules that become functional when folded into specific 3-D structures. In many cases the functions of these molecules are tied to their controllable motion properties. These natural molecular machines have optimally evolved to conduct very specific biological tasks in living organisms, but the natural "design" process is far from being understood well enough to be reproducible. Furthermore, the 20 amino acids of the standard genetic code are only a tiny fraction of the number of α-amino acid chemical structures that could not only play a role in the natural processes supporting human life, but also in the engineering of the artificial nano-machines of the future.

This thesis formulates a theoretical and computational framework to enable systematic explorations of the design space of self-assembled nano machines with prescribed mobility. One of the key differences between designing at the macro and nano scales is that, in the latter case, one does not have the freedom to fabricate "components" of desired shapes and sizes. Instead, the types of possible nano components that are either available or can be fabricated are finitely many. Therefore,

we propose a systematic strategy and computational infrastructure to design manufacturable molecular machines with prescribed mobility and function obtained from a predefined library of molecular components. Furthermore, we investigate the design space of one-degree of freedom (DOF) nano-machines, which are known to be the simplest, most effective, robust, and widely used designs at the macro-scale because of their completely predictable and repeatable motion.

The resulting synthesis procedure is the first of its kind and capable of synthesizing functional linkages with prescribed mobility constructed from a soup of primitive entities. The preliminary investigations have already led to the discovery of novel, never before seen one degree of freedom nano-machines, which have been proven both in simulations and experiments to self-assemble into one degree-of freedom nano-robots. Equally important, the proposed systematic approach can enumerate an ATLAS of candidate nano-mechanisms with prescribed mobility, and can be used by "nano-designers," e.g, synthetic chemists, biochemists, biologists, pharmacists, and engineers, to explore the vast design spaces of artificial molecular machines of the future, and develop, for example, novel drug delivery agents, nano-robots and sensors, as well as programmable matter.

On the Systematic Design and Analysis of Artificial Molecular Machines

Pouya Tavousi

B.S. Aerospace Engineering

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the

University of Connecticut

2016

APPROVAL PAGE

Doctor of Philosophy Dissertation

On the Systematic Design and Analysis of Artificial Molecular Machines

Presented by

Pouya Tavousi, B.S. AE.,

Co-Major Advisor _____
Horea Ilies

Co-Major Advisor _____
Kazem Kazerounian

Associate Advisor _____
Andrei Alexandrescu

Associate Advisor _____
Julian Norato

Associate Advisor _____
George Lykotrafitis

University of Connecticut

2016

ACKNOWLEDGMENTS

My sincere appreciation goes to my advisors, Prof. Kazem Kazerounian and Prof. Horea Ilies for their support and mentorship. I would like to also express my gratitude to Dr. Vitaliy Gorbatyuk, Prof. Sanguthevar Rajasekaran and Dr. Sharareh Emadi as well as my PhD committee members, Prof. George Lykotrafitis, Prof. Andrei Alexandrescu and Prof. Julian Norato for their great help. Special thanks to my colleagues at Kinematic Design Laboratory, Martin Huber, Matt Eschbach, Hima Khoshreza, Zahra Shahbazi and Hami Golbayani as well as Computational Design Laboratory, Ata Eftekharian, Morad Behandish, Radu Corcodel, Reed Williams, Frol Periverzov and Denis Dorozhkin for their useful feedback. I would like to also sincerely thank my awesome Iranian friends. I would like to extend my heartfelt thanks to my beloved family for their unconditional love and constant support, Mahmood, Zohreh, Anahita, Arash, Amir, Nasim and Radin, especially my parents that taught me everything and set me up for life.

Contents

Ch. 1. INTRODUCTION 1

Ch. 2. Synthesizing Functional Mechanisms From a Link Soup 10
2.1 INTRODUCTION . . . 10
2.2 METHODS . . . 16
2.2.1 Enumerating Topologies for Prescribed Mobility . . . 16
2.2.2 Generating Linkage Arrangements . . . 36
2.2.3 Geometry . . . 37
2.3 RESULTS . . . 45

Ch. 3. Systematic Design, Analysis and Control of Manufacturable Nano Machines 52
3.1 INTRODUCTION . . . 52
3.2 METHODS . . . 57
3.2.1 Input Preparation . . . 57
3.2.2 Design (Kinematic Synthesis) . . . 58
3.2.3 Control Strategies . . . 59
3.2.4 Fabrication and Validation . . . 61
3.2.5 Functionalization . . . 62
3.3 RESULTS AND DISCUSSION . . . 63

Ch. 4. PROTOFOLD: Protein Folding Prediction Package 78
4.1 Introduction . . . 78
4.1.1 Related Work . . . 79
4.1.2 Outline & Contributions . . . 83
4.2 Formulation . . . 85
4.2.1 Kinematic Model . . . 86
4.2.2 Force Model . . . 92

4.2.3 Kinetostatic Simulation . . . 101
4.3 Algorithms . . . 105
4.3.1 Rigid Transformations . . . 105
4.3.2 Proximity Queries . . . 106
4.3.3 Bonds Tree/Graph . . . 112
4.3.4 Surface Enumeration . . . 113
4.3.5 Prefix Sum Calls . . . 121
4.4 Implementation . . . 123
4.4.1 **Protofold II** Architecture . . . 123
4.4.2 CPU-Parallel Implementation . . . 129
4.4.3 GPU-Parallel Implementation . . . 131
4.5 Results & Discussion . . . 134
4.5.1 The Folding Process . . . 136
4.5.2 Computation Times . . . 143
4.5.3 Real Examples . . . 147

Ch. 5. Approximating Net Interaction Among Rigid Bodies, Resulting From Pairwise Interactions Between Their Constituents 156
5.1 INTRODUCTION . . . 156
5.1.1 Ignoring Far Interactions . . . 157
5.1.2 Using Mesh Values and Interpolation . . . 158
5.1.3 Exploiting Rigidity . . . 158
5.2 METHODS . . . 166
5.2.1 Separated Variable Representation . . . 166
5.2.2 Polynomial Regression/Expansion . . . 167
5.2.3 Error Analysis . . . 172
5.3 RESULTS AND DISCUSSION . . . 173
5.3.1 Electrostatic Energy of Two Rectangular Cubes, 1D Translation 173
5.3.2 Electrostatic Energy of Two Rectangular Cubes, 1D Rotation 177
5.3.3 Almost-Rigid Bodies . . . 177
5.3.4 Net Force and Net Moment Approximation . . . 178

Ch. 6. CONCLUSION 182

Ch. A. Peptide Chains 187

Ch. B. Prefix Computation 189

Ch. C. Parallel Computing 190
C.1 Abstract Machines . . . 190

C.2 GPU SIMT Model 191

Ch. D. Sifting the Conformation Space for More Efficient Sampling 192

Bibliography 209

Chapter 1

INTRODUCTION

Nature has built billions of different machines through evolution. On the other hand, humans have developed the design methodology to make machines. Although in many cases inspired by nature, the macro scale machine design method has unique features which make it a very popular alternative to the natural evolution for artificial machine synthesis: the designed machines have fewer number of components with respect to naturally existing machines and thus are simpler to build; the behaviours of the designed machines are easier to predict and thus these machine are easier to control; and yet, the designed machinery are capable of functions done by the naturally existing equivalents. A vivid example can be observed in the comparison between naturally evolved flying bird and its alternative, the designed airplane.

The story is however different at the molecular scale. Nature has manipulated the genetic data through natural evolution to assemble long protein chains out of only 20 different building blocks called amino acids. Each chain then folds into a specific three-dimensional functional form. This three-dimensional structure, also called the tertiary

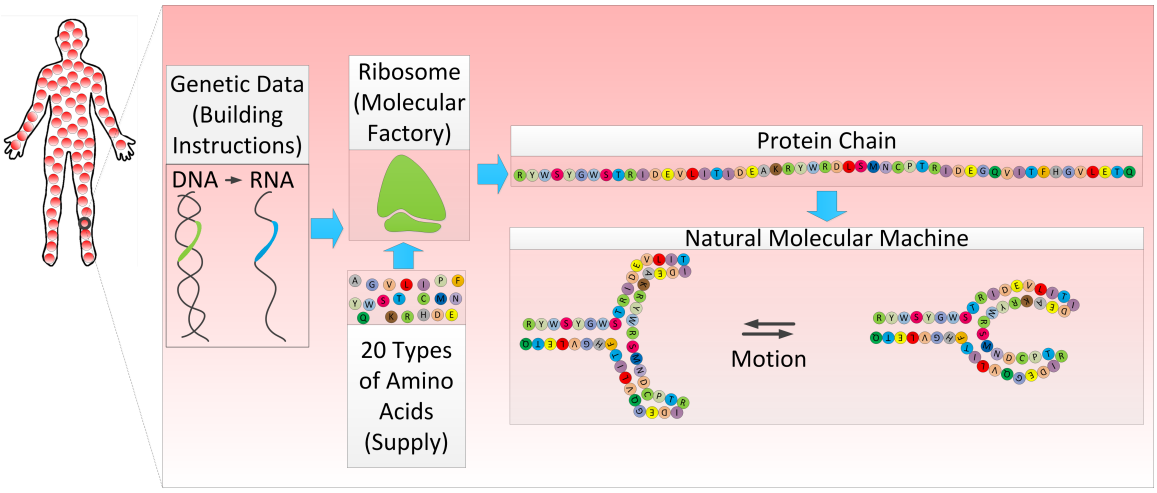


FIGURE 1.0.1: Natural molecular machines are products of evolution.

structure, is observed to be only a function of sequence, also called the primary structure [91]. Furthermore, in many cases the functions of these molecules are tied to their controllable motion properties (Fig. 1.0.1).

A wide range of applications can be thought for the molecular machines, form being used as inhibitory drugs for fighting lethal diseases such as Alzheimer [164] and cancer [54], to acting as bio-sensors [61] and bio-actuators [46]. The natural molecular machines have optimally evolved to conduct very specific biological tasks in living organisms. However, the 20 amino acids of the standard genetic code are only a tiny fraction of the number of α-amino acid chemical structures [110] that can potentially be used as building blocks of molecular machines. Moreover, many possible combinations of even these limited set are not currently seen in nature. This results in a molecular machine design space that is sparsely explored which motivates an effort for designing artificial molecular machines, capable of tasks that are not seen among natural versions of these machines (Fig. 1.0.2).

The most common current molecular design strategy attempts to mimic nature

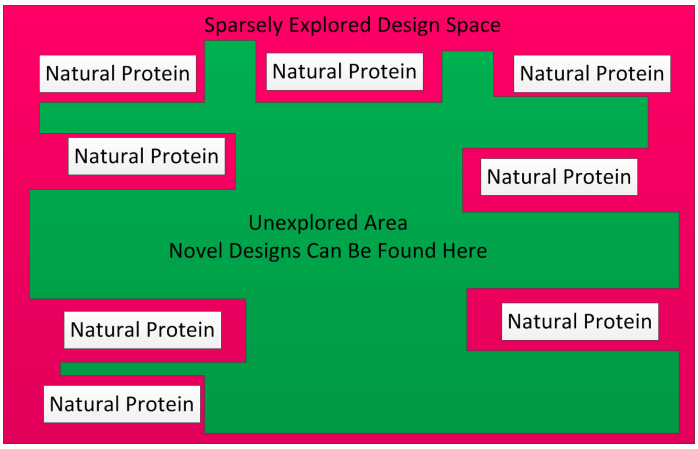


FIGURE 1.0.2: Sparsely explored design space.

in the way that it tries to reverse engineer the natural folding process (Fig. 1.0.3) to make new functional machines ([25, 57, 58, 95, 105, 124, 154]). For this method to be efficient, one needs to have a very good understanding of the folding process. Given the fact that experimental setup for analyzing the protein molecules takes a lot of effort, time and expense, to widely study the behaviour of a large number of molecular machines, there is a tendency towards computer simulations. A key factor in developing a computer simulation is to pick up a reasonable balance between accuracy of the physical model and the computational efficiency. Making use of laws of quantum mechanics, one can predict the behaviour of protein systems with high accuracies [101]. However, since usually thousands of atoms are engaged in such systems, using quantum models is almost impractical from a computational perspective. In fact it seems that, given today's existing computational power, there is no option but to compromise accuracy to gain computational efficiency for the simulation times to be reasonable. Unfortunately, due to lack of good understanding of the folding process, current molecular simulation tools suffer from poor models that are not sufficiently fast or accurate. As a result, analysis of molecular systems and

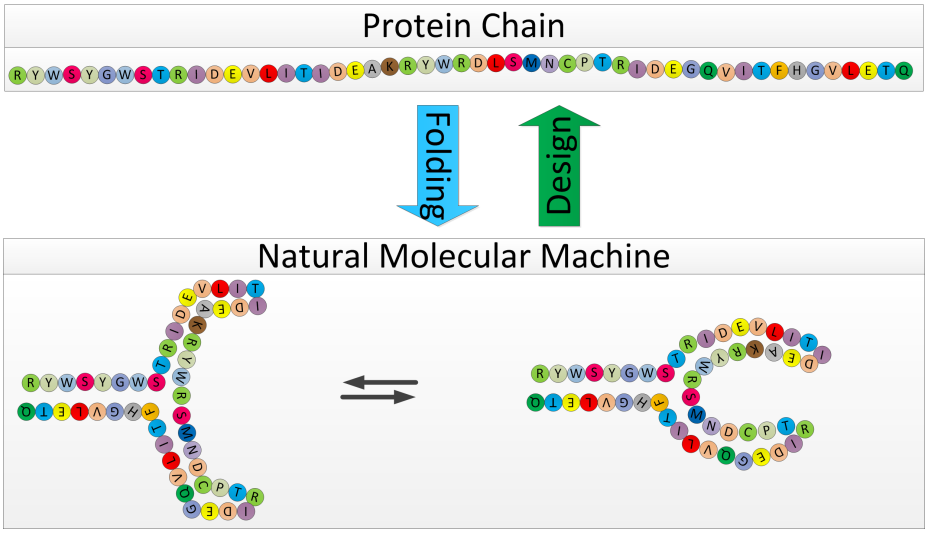


FIGURE 1.0.3: Reverse engineering the protein folding process for protein design.

consequently, design through the reverse process are complex. Besides, prediction and thus controlling the motion of the designed machines is an even harder task due to the several competing factors that affect the molecular motion. Especially, when the size grows, the trade-off between computational efficacy and the accuracy of modeling places real challenges on the way of the designer.

In another approach, synthetic chemists and biochemists, in a rather trial and error fashion, try to apply their knowledge and experience to perform nano-design similar to what is practiced in the macro scale [4, 12, 13, 16, 22, 29, 78, 79, 81, 89, 115, 119, 153, 183] (Fig. 1.0.4). The most critical challenge faced in these ad hoc techniques is that making nano building block components at the arbitrary shapes and sizes is very restricted. This means that one must use the naturally existing structural entities or the ones that can be artificially synthesized. Furthermore, governing physics at the nano scale are experienced differently from the macro scale. Therefore, motion design and control strategies practiced in the macro scale cannot be directly used at

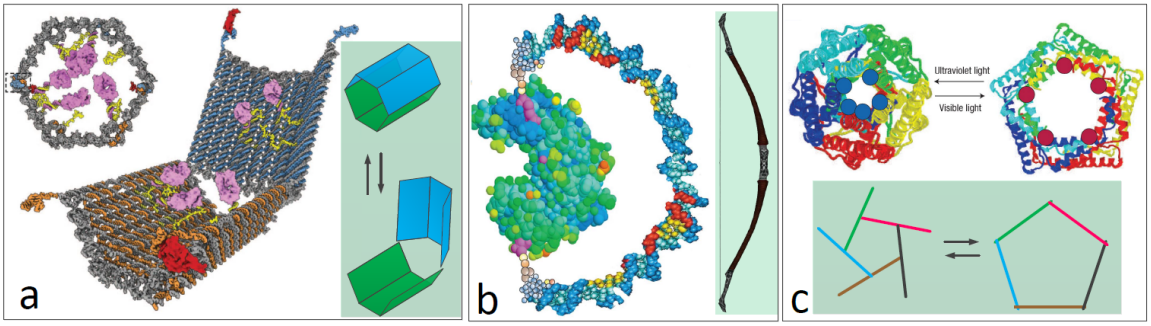


FIGURE 1.0.4: Examples of existing ad-hoc designs of nano machines: (a) aptamer-gated DNA nanorobot in open and closed configuration with protein payload [44]; (b) applying mechanical tension on a protein via a "molecular spring" [33]; (c) a light actuated nano-valve [22].

the molecular level.

It seems that a method than can systematically search for controllable molecular machines is lacking. Looking at the existing molecular machinery, it is realized that, an analogy exists between theses machine and the mechanical mechanisms. For instance, the classical machine components such as kinematic links and joints can be found at the molecular level as well. In this thesis, we seek to answer the following question: can we bring together the rich knowledge and the well-established methods of mechanical design, structural biochemistry and computer science to tailor the conventional macro scale design methods to enable systematic exploration of the design space of controllable molecular machines? (Fig. 1.0.5)

In addition, we will attempt to develop more powerful protein folding prediction techniques, in terms of the balance between accuracy and computational efficiency, to be used towards more reliable reverse-folding protein design. We will also try to develop general novel computational techniques that facilitate analysis and design of molecular systems and can be applied to a wide range of molecular simulation applications.

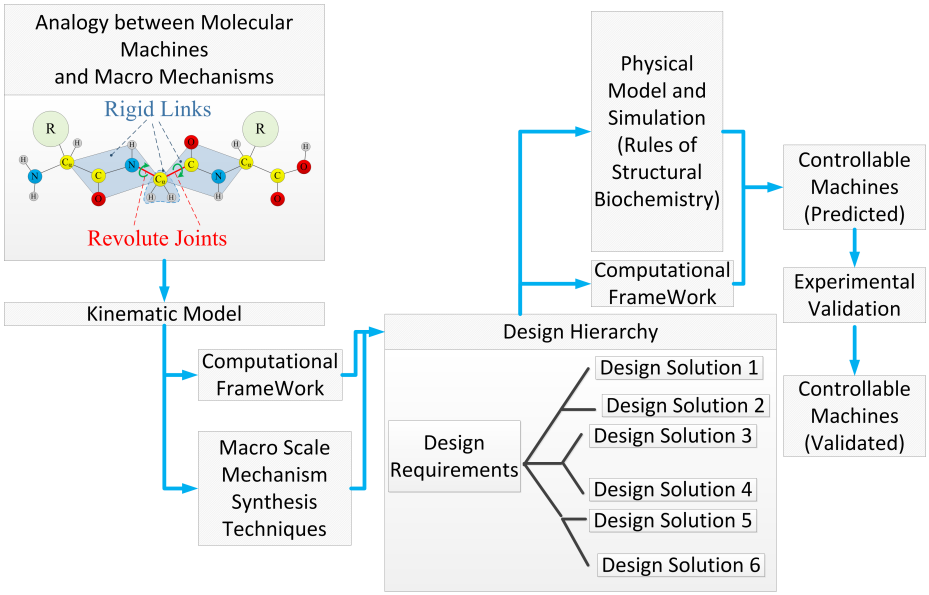


FIGURE 1.0.5: Systematic search for controllable molecular machines

In chapter 2, we will try to address the constraints in synthesis of functional molecular machines imposed by difficulties in fabricating nano-links of arbitrary shapes and sizes [162]. Classical mechanism synthesis methods, which assume the ability to manufacture any designed links, and therefore, are unable to provide a systematic process for assembling linkages from a limited set of links, will be tailored. We will propose a new approach to build functional mechanisms *with prescribed mobility* by only using elements from a predefined "link soup". First, we enumerate an exhaustive set of topologies, while employing divide-and-conquer algorithms to control the generation and elimination of redundant topologies. Then, we construct the linkage arrangements for each valid topology. Finally, we output a set of feasible geometries through a positional analysis step that minimizes the error associated with closure of the loops in the linkage while avoiding geometric interference. The proposed systematic approach outputs the ATLAS of candidate mechanisms, which can be further pro-

cessed for downstream applications. The resulting synthesis procedure will be the first of its kind that is capable of synthesizing functional linkages with prescribed mobility constructed from a soup of primitive entities.

In chapter 3, we will apply the proposed methods in chapter 2 to molecular components to give rise to a systematic and generic strategy for design, analysis and control of manufacturable molecular machines. Based on the principles of classical mechanical design, "design for function" at the molecular level can be enabled by first putting together irreducibly simple machines out of a set of available molecular components to yield constrained and consequently controllable motions (preferably one-degree of freedom), followed by devising control mechanisms to manipulate the resulting motions, and finally functionalizing the fabricated machines, to be exploited in different applications individually or as parts of an ensemble. Design and simulations are conducted through our developed computational framework and validations are performed via various physical tests. Our preliminary physical tests support the formation of a one-degree of freedom cyclic pentapeptide as speculated by our simulations. Furthermore, highly promising design cases can be collected in a nano-machinery ATLAS which acts as a powerful resource for synthetic chemists, biochemists, biologists and pharmacists and narrows down the vast and complex design space of molecular machinery design.

Chapter 4 is dedicated to our contribution to protein folding prediction problem [159]. We will enhance the previously developed kinetostatic compliance method (KCM) implemented into the Protofold package that was shown to overcome some of the key difficulties faced by other *de novo* structure prediction methods, such as the very small time steps required by the molecular dynamics (MD) approaches or the very large number of samples needed by the Monte Carlo (MC) sampling tech-

niques. We will improve the free energy formulation used in Protofold by including the typically underrated entropic effects, imparted due to differences in hydrophobicity of the chemical groups, which dominate the folding of most water-soluble proteins. In addition to the model enhancement, we revisit the numerical implementation by redesigning the algorithms and introducing efficient data structures that reduce the expected complexity from quadratic to linear. Moreover, we develop and optimize parallel implementations of the algorithms on both central and graphics processing units (CPU/GPU) achieving speed-ups up to two orders of magnitude on the GPU. Our simulations are consistent with the general behavior observed in the folding process in aqueous solvent, confirming the effectiveness of model improvements. We will report on the folding process at multiple levels; namely, the formation of secondary structural elements and tertiary interactions between secondary elements or across larger domains. We also observe significant enhancements in running times that make the folding simulation tractable for large molecules.

With the emergence of high computational power, the zeal for sophisticated approximation methods that can formulate the problem with smaller number of variables has decreased. For a large portion of simulation practices, this is compensated easily by the computational power. But, for molecular systems that consist of thousands of entities and must be studied in rather long timescales, the story is different. Therefore, the need for developing novel efficient approximation techniques that account for the special attributes of the problem at hand is strongly felt. We will try to make contribution to this, in chapter 5. Many physical simulations aim at evaluating the net interaction between two rigid bodies, resulting from the cumulative effect of pairwise interactions between their constituents. Examples of this phenomenon can be seen in hierarchic protein folding instances where, the interaction between almost

rigid domains directly influences the folding pathway [11], interaction between macro-molecules [77, 108, 181] for drug design purposes, self-assembly of nano-particles [186] for drug design and drug delivery applications, design of smart materials [60] and bio-sensors [102], as well as the interaction between stellar clusters [51]. In general, the brute force approach requires quadratic (in terms of number of particles) number of pairwise evaluation operations for any relative pose of the two bodies, unless simplifying assumptions lead to a collapse of the computational complexity. We propose to approximate the pairwise interaction function using separated variables method in order to split the variables that describe local geometries of the two rigid bodies and the ones that reflect the relative pose between them. Doing so replaces the quadratic number of interaction evaluations for each relative pose with a one-time quadratic computation of a set of characteristic parameters at a preprocessing step, plus constant number of pose function evaluations at each pose, where this constant is determined by the required accuracy of approximation as well as the efficiency of the used approximation method. We will show that the standard deviation of the error for the net interaction is linearly (in terms of number of particles) proportional to the regression error, if the regression errors are from a normal distribution. Our results show that proper balance of the tradeoff between accuracy and speed-up yields an approximation which is computationally superior to other existing methods while maintaining reasonable precision. Moreover, the decomposition of pose and local geometry facilitates the design process of desired behaviours in a system of rigid bodies.

Chapter 2

Synthesizing Functional Mechanisms From a Link Soup

2.1 INTRODUCTION

The classical approach to mechanism synthesis involves three steps: (a) *type synthesis* to select the type of mechanism realizing a specific motion requirement; (b) *number synthesis* to determine the topology of the mechanism fulfilling the prescribed mobility requirements; and (c) *dimensional synthesis* to specify the geometric dimensions of the synthesized mechanism [166]. One of the key assumptions of this classical synthesis procedure is that the resulting links output by step (c) can be manufactured to the designed dimensions.

However, fabricating links at the nanoscale, for example as needed by molecular machines, is notoriously difficult [150] and the shapes and sizes of the links that can be potentially manufactured in a lab environment are limited. In fact, the most vi-

able approach currently available for synthesizing nano-mechanisms is to use existing molecular components (e.g. rigid fragments of amino acids) as building blocks for constructing new molecular machinery. Therefore, a feasible molecular mechanism synthesis strategy can only use a finite number of types of links that either exist in nature or could be fabricated. This restriction forces the replacement of the *dimensional synthesis* step of the traditional method with the *positional analysis* stage, which investigates the closure of kinematic loops. It also reformulates the design task into seeking functional mechanisms that can be assembled by only picking elements from a *link soup* of primitive entities. We note that this new design task should result in an ATLAS of candidate mechanisms, which are kinematically plausible.

In this chapter, we propose a linkage synthesis method (Fig. 2.1.1) aimed at supporting mechanism synthesis at the nano-scale for a prescribed number of degrees of freedom (DOF). The proposed method:

1. *enumerates all valid topologies* that satisfy certain mobility conditions in terms of the desired DOF of the final mechanism;

2. generates all possible *linkage arrangements* by assigning links from the link soup to each topology, and

3. performs *positional analysis* for every *linkage arrangement*, to extract the geometrically feasible solutions.

The topology enumeration phase of the traditional mechanism synthesis aims to produce topologies represented by graphs, followed by detecting cases of isomorphism to yield a canonical set of non-redundant solutions satisfying prescribed topological criteria [166]. However, graph isomorphism detection is an NP-complete problem

[111]. Comprehensive reviews of the conventional methods for topology enumeration and isomorphism detection in the context of mechanism synthesis are provided in [24, 113], and a recent review of the graph matching literature appears in [52]. In general, the task of discarding redundant topologies requires a quadratic number of pairwise comparisons. A different approach is taken in [184] to avoid the generation of redundant topologies based on permutation groups. However, this technique cannot address the redundancies of linkage arrangements, i.e. when graph vertices carry additional information about the link properties as well as the joint connectivity of adjacent links. Furthermore, this method enumerates topologies containing only single edges, which can become problematic when multiple connections need to be formed among constituents, a common occurrence at the molecular level, where the building blocks and the interactions between them are more intricate than at the macro scale. One example of such a case is presented in [104] showing two molecular domains connected by multiple hydrogen bonds.

We propose a generic technique for generating non-isomorphic graphs with prescribed topological characteristics (Fig. 2.1.1). The proposed divide and conquer method controls the redundant topologies that are generated, which reduces the overall number of required isomorphism checks. By detecting and eliminating the graph redundancies as soon as they emerge, we reduce the exponential growth to linear growth. Our method first generates the *link families*[1] that are associated with the prescribed DOF [166], and that satisfy prescribed bounds on the number of links or connections/joints for each link selected from the link soup. Then, for each link family, we enumerate all feasible topologies as discussed in section 2.2. The hierarchical structure induced by the divide and conquer approach can not only control the

[1]We use the term *link family* according to the definition provided in [166].

propagation of redundant topologies, but may also be used to store the link families corresponding to a prescribed mobility as well as the topologies belonging to a link family. These solutions can then be retrieved and reused efficiently as needed during the current or subsequent design processes. Note that although the proposed computation of the link families, as an intermediate enumeration product, expedites the topology enumeration process, it can also produce topologies with undesired DOF due to the emergence of over-constrained subgraphs [171]. Thus, the topology enumeration step is followed by a mobility analysis[2] as a post-processing step. In this chapter, we use the method presented in [146] to identify the over-constrained subgraphs, and replace them with rigid links, although we note that other approaches to detect anomalies in local mobility have been documented in the literature [43, 149] [3].

In principle, any well-established isomorphism detection technique can be used to test for redundancies at each level of the divide-and-conquer hierarchy. However, we propose a new heuristic similarity detection technique that eliminates the pairwise-comparison between topologies by assigning a characteristic matrix to every topology as discussed in section 2.2, leading to a more computationally efficient process. This characteristic matrix acts as a "topological signature," and allows us to evaluate the presence of isomorphisms in a set of candidate topologies. We illustrate the efficiency of the proposed isomorphism detection for our problem through the benchmark tests discussed in section 2.3. A slightly modified version of the similarity detection method is also used to identify similarities during the generation of linkage arrangements (Fig.

[2]See also [65].

[3]Identifying overconstrained regions is a relatively minor task in the proposed enumeration process, and, in practice, its efficiency has a relatively small impact on the overall computational efficiency. The method used here has been developed by this research group, which allowed us to rapidly develop the implementation. The authors, however, acknowledge the existence of more efficient techniques, which can be used in subsequent developments.

2.1.1), when link components are assigned from the link soup to the vertices of the candidate topologies.

The final step of the proposed approach is the positional analysis, when the joint variables in the linkage arrangements are adjusted to satisfy two sets of requirements: (1) simultaneous closure of kinematic loops (Fig. 2.1.1), and (2) no geometric interference. This step can be formulated as an optimization problem, and any traditional optimization method proposed in the literature [27, 85, 100, 158] can in principle be used. In section 2.2 we describe one specific implementation of an optimization algorithm, which takes advantage of the particular attributes of the method that we propose. The objective function is defined as an error function that takes into consideration the kinematic loop closure as well as the amount of geometric overlap between the links.

To the best of our knowledge, the resulting tool is the first synthesis tool aimed at finding linkages that can be constructed from a soup of primitive entities that may be available to the designer. Although the scope of this technique is rather general, its immediate application is the design of molecular machines which have to be assembled from nano-links that either exist or can be manufactured at that scale. As is commonly the case, molecular designers can use their knowledge of chemistry and biochemistry to compile a set of chemical building blocks that can be considered to be rigid links and to form the link soup. We illustrate the effectiveness of the approach by reproducing the regular typologies reported in [109], and its efficiency by providing a comparison with a traditional isomorphism detection approach. We also provide a validation test for the positional analysis implementation. Finally, we used the synthesis tool to design a one DOF 7-bar protein-based linkage from existing molecular components and to compute its range of motion. The divide-and-conquer method

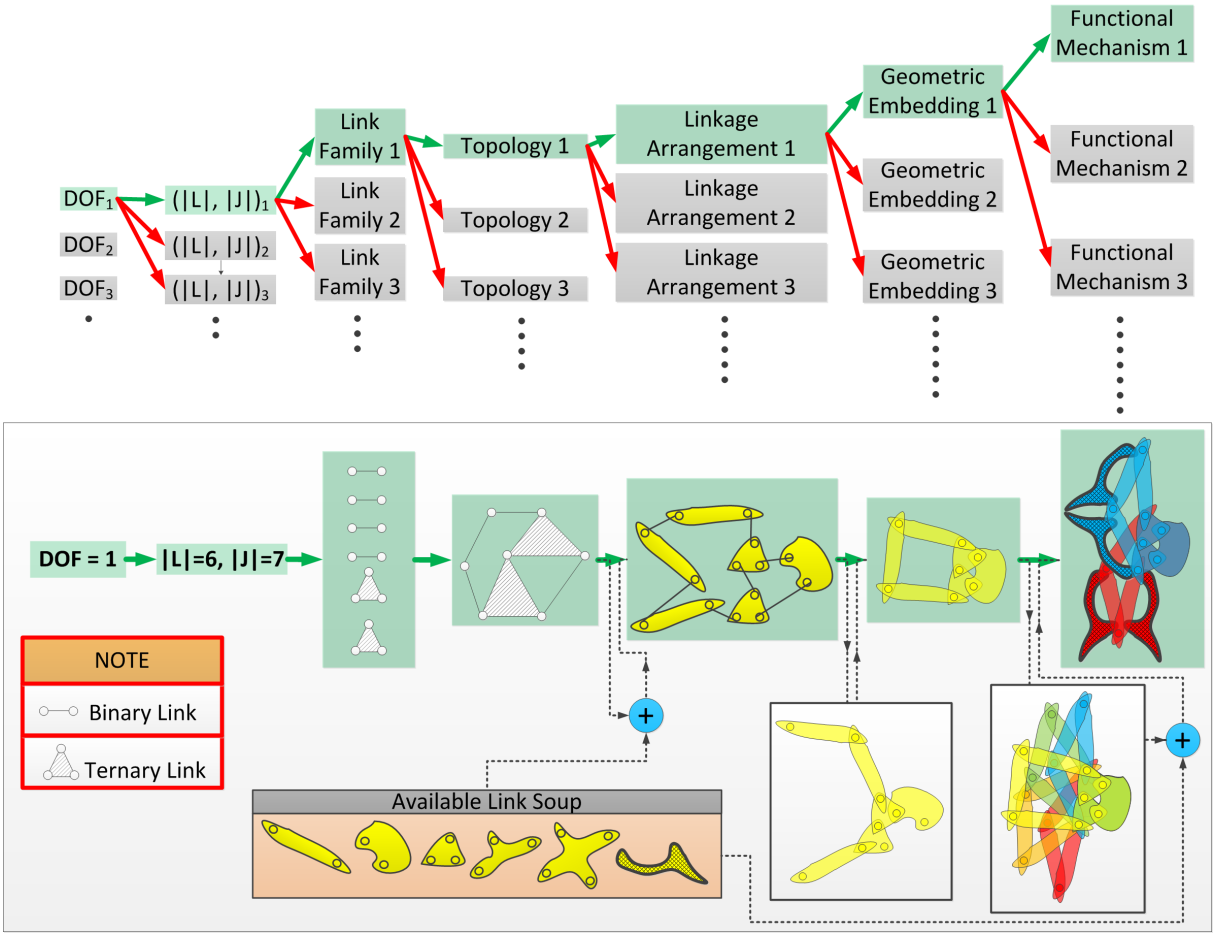


FIGURE 2.1.1: The proposed mechanism synthesis procedure.

proposed here augmented by the heuristic graph isomorphism detection technique results in a significantly faster synthesis process than the conventional mechanism synthesis methods. Furthermore, the two step redundancy elimination (redundant topologies and redundant linkage arrangements) has a critical effect in improving the computational efficiency during the positional analysis phase. Finally, the specific formulation and implementation of the error minimization during positional analysis leads to an effective functional mechanism synthesis procedure for a prescribed mobility requirement.

2.2 METHODS

Figure 2.1.1 provides a bird's eye view of the proposed synthesis technique that outputs the ATLAS of candidate mechanisms along with their ranges of motion. The methods presented in this section are designed and developed to address the specific nano-design problem at hand. Our focus was on rapid development of a robust and practical design tool, but fine-tuning its computational efficiency may require further investigation of other implementations for solving specific tasks.

2.2.1 Enumerating Topologies for Prescribed Mobility

Mobility, which by definition captures the degrees of freedom or, alternatively, the number of independent variables needed to completely describe the configuration of a linkage [166], acts as one of the main inputs for our synthesis tool. Given the geometry of a mechanism, one can easily determine its associated DOF. However, the inverse problem is critical in the mechanism synthesis process. The Grübler-Kutzbach criterion suggests a formula for quick determination of the DOF based only on the knowledge of the number of links and number and types of constraints in the mechanism. It must be noted however, that due to the existence of over-constrained regions in some mechanisms, the suggested formulae may report mobilities that are lower than the true values. In other words, Grübler-Kutzbach yields only a lower bound for the DOF of a mechanism [142]. Here, we use this criterion as a "bridge" between mobility and topology. Specifically, assuming that the criterion reports the correct value of DOF for a mechanism, an initial set of topologies whose mobility is at least the desired DOF value is generated. This is followed by a mobility analysis of the resulting topologies to filter out the candidates that contain over-constrained

regions.

Without loss of generality, we account only for revolute and prismatic joints, since other joints can be replaced by a combination of joints from these two types. Under these assumptions, the Grübler-Kutzbach criterion becomes

$$DOF = \lambda\,|L| + (1-\lambda)\,|J| - \lambda. \tag{2.2.1}$$

In this equation, L and J are respectively the sets of links and joints and $|X|$ reflects the cardinality of a set X. Parameter λ indicates the DOF of a single rigid link before its motion is constrained. This number equals 3 for planar and spherical mechanisms, and 6 for spatial linkages.

Equation (2.2.1) yields the degrees of freedom of a given topology. To address the inverse problem, we first find a list of *link families* that satisfy equation (2.2.1), and then enumerate topologies for each family. By definition, a link family or *link assortment* is a tuple of numbers that indicate how many links of each type are present in the mechanism, so a link family represents multiple topologies. Let $L_i \subset L$ represent the set of links in a mechanism with degree i, or, alternatively, with i attachment nodes. An example of a link family in the plane is $(|L_2| = 4, |L_3| = 2)$, which represents all the 6-bar planar mechanisms that have 4 binary and 2 ternary links.

Enumerating Link Families For a Given DOF

Given a DOF we construct the link families in two steps. First, we determine all possible pairs of $(|L|, |J|)$ that satisfy equation (2.2.1) by considering that $|L|$ and $|J|$ must be integers, which implies that equation (2.2.1) can be rewritten as:

$$|L| = (\lambda - 1)k + DOF + 1 \tag{2.2.2a}$$

$$|J| = \lambda k + DOF \tag{2.2.2b}$$

where k can be any non-negative integer number. Next, for each pair $(|L|, |J|)$ we generate a set of link families such that the following two equations hold:

$$|L| = \sum_i |L_i| \tag{2.2.3a}$$

$$2|J| = \sum_i i \cdot |L_i| \tag{2.2.3b}$$

where $|L_i|$ is the number of links with degree i. In order to generate the link families, equations (2.2.3a) and (2.2.3b) must be solved simultaneously, which can be viewed as determining different $|L_i|$ values. This is accomplished by recursively breaking the problem into two subproblems: (1) selecting the $|L_i|$ value for the smallest i (2) selecting the rest of $|L_i|$ values. The base case in the recursion happens when only one $|L_i|$ remains to be selected, which is trivial.

For example, consider the case of planar and spatial mechanisms with $DOF = 1$, which contain up to 8 links. The following $(|L|, |J|)$ pairs will simultaneously satisfy equations (2.2.2a) and (2.2.2b): $\{(|L| = 2, |J| = 1),(|L| = 4, |J| = 4),(|L| = 6, |J| = 7),(|L| = 8, |J| = 10)\}$ for planar and $\{(|L| = 2, |J| = 1),(|L| = 7, |J| = 7)\}$ for spatial mechanisms. For the $(|L| = 6, |J| = 7)$ pair, it is known that $|J| - |L| + 2$ is an upper bound for the number of nodes on each link of a generic mechanism, which equals 3 in this case. Therefore, the highest order link would be ternary, and equations (2.2.3a)

and (2.2.3b) become:

$$6 = |L_1| + |L_2| + |L_3| \tag{2.2.4a}$$

$$14 = |L_1| + 2|L_2| + 3|L_3| \tag{2.2.4b}$$

By assigning an integer value to $|L_1|$ between 0 and 2, equations (2.2.4) become a system of two equations with the two unknowns $|L_2|$ and $|L_3|$, which will produce three families: $\{(|L_1| = 0, |L_2| = 4, |L_3| = 2), (|L_1| = 1, |L_2| = 2, |L_3| = 3), (|L_1| = 2, |L_2| = 0, |L_3| = 4)\}$. Note that assigning other values to $|L_1|$ results in invalid answers (i.e. negative $|L_i|$'s).

Enumerating Topologies For Each Link Family

We can now proceed to enumerate topologies that belong to each family. In the graph representation of the linkages, vertices and edges correspond to links and joints, and the degree of a vertex specifies the type of the link. Also, each vertex or edge has an assigned label. Two topologies that result from *different* label assignments of the *same* connectivity between links are called isomorphic. It is worthwhile noting that topologies produced from different families cannot be isomorphic. However, topologies that are generated from the same family are often isomorphic, so by eliminating isomorphic topologies during the enumeration one would dramatically reduce the number of computations required during the later stage of this synthesis process. We address this problem by using a divide-and-conquer algorithm with extensive branching, and aim to fragment the problem into the largest possible number of non-overlapping subproblems as discussed next.

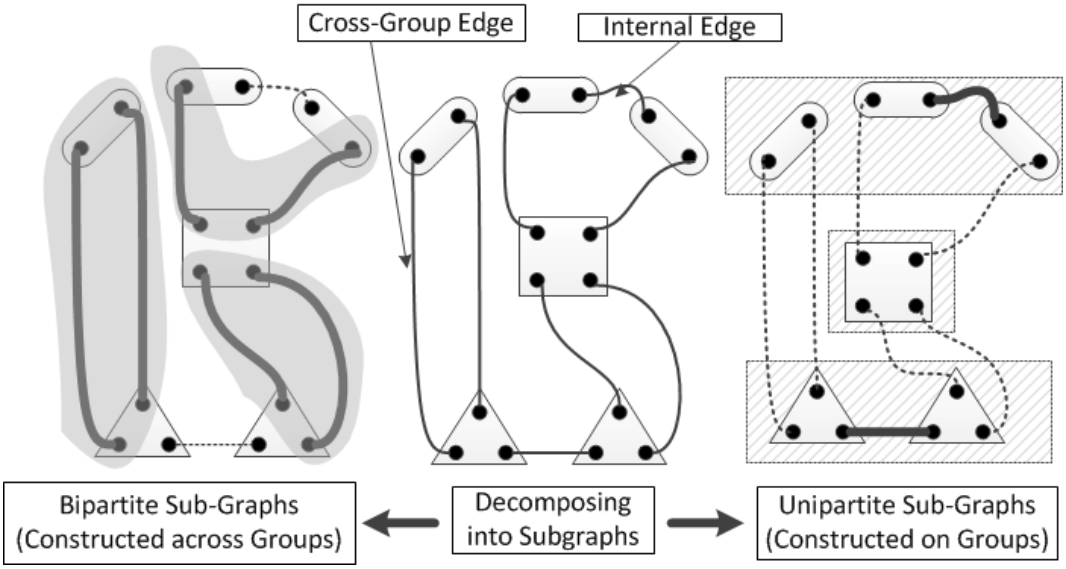


FIGURE 2.2.1: Vertices are grouped by their degrees and the graph is decomposed into two sets of subgraphs: the *unipartite* subgraphs formed within a degree group, and the *bipartite* subgraphs formed across degree groups.

Clustering Graph Vertices By Their Degree Given a link family, the first step is to cluster the graph vertices by their degree into the so called *degree groups*. Enumerating all graphs belonging to a link family is equivalent to finding all possible ways that connections (edges) may be formed between vertices of the graph such that the degrees of all vertices are matched. At this point, we introduce a decomposition of the full graph into two different sets of subgraphs: the *unipartite* subgraphs, constructed only with the vertices belonging to one degree group (Figure 2.2.1), and the *bipartite* subgraphs formed by connecting vertices from two different degree groups (Figure 2.2.1). Taking a divide-and-conquer approach, we enumerate the resulting graphs in two steps: first, we find all the different ways in which the edges can be distributed among the subgraphs; then we perform the graph enumeration for each case.

Distributing Edges Among Subgraphs Figure 2.2.2 demonstrates the *coarse adjacency matrix* of a generic graph for which the vertices are partitioned into n groups based on their degrees. Let element $\bar{a}_{ij}$, represent the number of connections between vertices of group i and group j. When $i = j$, $\bar{a}_{ii}$ equals twice the number of the internal edges in group i. Finding different edge distributions among the subgraphs is equivalent to producing different allocations of positive numbers to the elements of this matrix in such a way that the three following conditions hold: (1) the summation of elements of row i equals the summation of degrees of vertices in group i (2) the matrix is symmetric, and (3) the numbers allocated to the diagonal elements are even. Thus, we can generate matrices satisfying these conditions in a straightforward divide-and-conquer fashion. At each level, the problem of specifying the matrix elements is partitioned into two subproblems: assign values to the first row (column), and allocate numbers to the submatrix resulting from the removal of the first row and the first column of the original matrix. The base case happens when the matrix carries only one element, when the assignment becomes trivial. Completing a row is also performed using a divide-and-conquer method where a row is split into its first element and a sub-row with the remaining elements in the original row. The base case here happens when the row has only a single element.

As an example, consider one of the previously identified link families: ($|L_1| = 0, |L_2| = 4, |L_3| = 2$). Since $|L_1| = 0$, the vertices are grouped into two classes corresponding to binary and ternary links. The coarse adjacency matrix is a 2×2 matrix with three independent elements: (1) $\bar{a}_{11}$ being twice the number of edges internal to the group of binary vertices (2) $\bar{a}_{12}$ or equivalently $\bar{a}_{21}$ being the number of cross edges between binary and ternary vertices (3) $\bar{a}_{22}$ being twice the number of edges internal to the group of ternary vertices. Now, we must assign values to

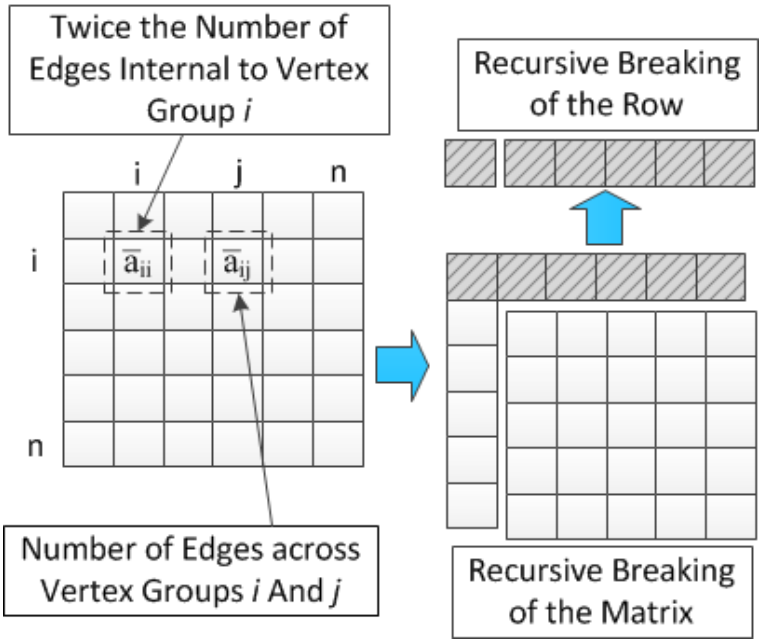


FIGURE 2.2.2: Different edge distributions among the subgraphs are computed in a divide-and-conquer fashion to specify the coarse adjacency matrix.

these three elements. To insure that the assigned values produce a valid solution, two conditions must be satisfied: $\bar{a}_{11} + \bar{a}_{12} = 8$ and $\bar{a}_{12} + \bar{a}_{22} = 6$. As described earlier, we proceed by: (1) specifying the elements of the first row, namely, allocating numbers to $\bar{a}_{11}$ and $\bar{a}_{12}$, and (2) by specifying the elements of the submatrix obtained by removing the first row and column, namely, allocating a number to $\bar{a}_{22}$. The first sub-problem, itself is broken down into selecting a value for $\bar{a}_{11}$ and then for $\bar{a}_{12}$. For instance, let's have $\bar{a}_{11} = 2$. This will imply $\bar{a}_{12} = 6$, which in turn yields $\bar{a}_{22} = 0$. The other valid solutions are $(\bar{a}_{11} = 4, \bar{a}_{12} = 4, \bar{a}_{22} = 2)$, $(\bar{a}_{11} = 6, \bar{a}_{12} = 2, \bar{a}_{22} = 4)$ and $(\bar{a}_{11} = 8, \bar{a}_{12} = 0, \bar{a}_{22} = 6)$.

Setting the Elements of the Fine Adjacency Matrix Although the coarse adjacency matrix informs us about the number of edges in different subgraphs, it does

not specify the vertices defining these edges within each of these subgraphs. Therefore, the next step in the graph enumeration process is to capture all different patterns for formation of these subgraphs, given a coarse adjacency matrix. This is equivalent to producing different combinations of elements in the *fine adjacency matrix*. Figure 2.2.3 shows the fine adjacency matrix for a generic graph. In this matrix, element of row k and column l, a_{kl}, corresponds to the number of edges drawn between vertices k and l. A fine adjacency matrix represents a valid graph if the following criteria are met: (1) the summation of the elements of row k must equal the degree of vertex k, and (2) the matrix must be symmetric. We accomplish the graph enumeration as follows: (a) on each vertex group i, we construct all possible unipartite subgraphs that carry $\bar{a}_{ii}/2$ edges, which is equivalent to listing all the solutions for the square submatrices placed on the diagonal of the fine adjacency matrix, as illustrated in Figure 2.2.3; it should be noted that the solution for each of these subgraphs is independent from the solution for any other subgraph at this step; and (b) for each two groups i and j, we form bipartite graphs across their vertices with $\bar{a}_{ij}$ edges. This is equivalent to listing solutions for the remaining rectangular submatrices in the fine adjacency matrix. In contrast to the previous step, the solution for each of these subgraphs is impacted by the solution of other subgraphs in this step and those from the previous step. In the following, we elaborate each of these steps in detail.

As an example, consider one of the coarse adjacency matrices we listed above for a 6-bar linkage: $(\bar{a}_{11} = 4, \bar{a}_{12} = 4, \bar{a}_{22} = 2)$. Let v_1, v_2, v_3 and v_4 represent the binary vertices and v_5 and v_6 be the ternary vertices. The task of building graphs for the given coarse adjacency matrix is split into generating instances for three subgraphs: (1) those on $\{v_1, v_2, v_3, v_4\}$ with exactly 2 edges, such that the maximal degree of any vertex is 2; (2) subgraphs on $\{v_5, v_6\}$ with exactly 1 edge such that the maximal

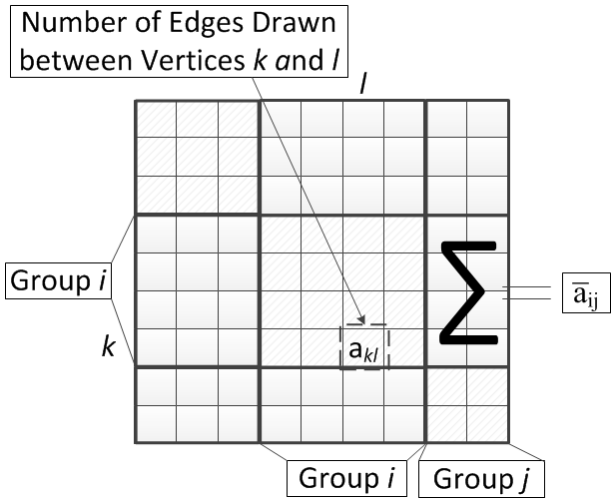


FIGURE 2.2.3: Graph enumeration is equivalent to finding solutions for the fine adjacency matrix. The task is split into generating two sets of subgraphs: the ones that are constructed on the vertices of the unipartite subgraphs, which correspond to square submatrices along the diagonal of the fine adjacency matrix, and the bipartite subgraphs, which correspond to the remaining rectangular submatrices. Note that the summation of elements in the submatrix associated with the intersection of groups i and j reflects $\bar{a}_{ij}$.

degree of any vertex is 3, and (3) bipartite subgraphs across the two groups with exactly 4 edges such that the maximal degree of any vertex from the first group is 2 and the maximal degree of any vertex from the second group is 3. Note that degrees are counted by accounting for edges that are formed in this step and also in the two previous steps.

Similarity in Graphs and Isomorphism Detection Before we proceed to detailing the steps of graph enumeration or equivalently generating adjacency matrices, we need to review the concept of vertex *similarity* in graphs. We use the concept of vertex similarity to infer the occurrence of graph isomorphism in different levels of recursion. Note that the sooner the isomorphism is identified in the recursion, the larger the eliminated chunks of the design space would be, leading to a more efficient

synthesis procedure.

Two vertices u and v in a graph G are similar if some automorphism of G maps u onto v [64]. In other words, two similar vertices are indistinguishable up to their labels. Vertex similarity is also directly related to the concept of graph isomorphism. For any two isomorphic graphs, one can find a one-to-one similarity relation between the sets of vertices and edges of the two graphs. In other words, detecting similarities in graphs can be used to detect or even avoid isomorphism in our graphs.

We know that different assignments of labels to one network of vertices results in isomorphic topologies. Equivalently, isomorphic topologies can be generated by shuffling the appropriate rows and columns of the corresponding fine adjacency matrix. This fact can be used in detecting similarity or isomorphism by simply examining different valid rearrangements of rows and columns in the adjacency matrices. Nevertheless, the computational complexity of doing so exhaustively can be significant. Instead, we offer a heuristic method for detecting similarities in the graph, which collects specific topological information about vertices and vertex pairs. These values are collected in a symmetric matrix, the so-called *characteristic matrix*, in which the off-diagonal element $n_{i,j}$ captures information about the topological 'relationship' between vertices i and j, and the diagonal element $n_{i,i}$ reflects the topological characteristic of vertex i in the graph. Importantly, if two vertices correspond to the same diagonal value in this matrix, they will be declared similar. In addition, if two pairs (v_1, v_2) and (v_3, v_4) correspond to the same off-diagonal value, then the topological 'relationship' between the first pair of vertices is similar to that of the second pair. To populate the characteristic matrix, we iteratively update its elements a predefined number of times, and our experience shows that 3 updates are sufficient.

The two key components of this process are: (1) a set of topological variables

that are easy to evaluate at every stage, such as degree of a vertex, the number of neighbors of a vertex, the common neighbors between two vertices, as well as the value of the characteristic matrix elements from the previous iteration; and (2) a real-valued characteristic function that takes these topological variables as input and outputs a number that is the same for similar vertices or vertex pairs, and is different for dissimilar vertices or vertex pairs. The characteristic matrix is initialized to zero, then iteratively updated by re-evaluating this characteristic function for every element of the matrix. As an example, consider the graph shown in Fig 2.2.4 along with the characteristic matrix determined with this heuristic method. The numeric values have been replaced with letters in this example to simplify the observation of similarities. The diagonal of the characteristic matrix suggests that vertices are grouped into three similarity categories: $\{v_1, v_3\}$, $\{v_2, v_4\}$ and $\{v_5\}$, which can be intuitively observed by looking at the graph. In addition, the off-diagonal values of the characteristic matrix inform us that all 10 vertex pair relations fall into 6 different categories: $\{(v_1, v_2), (v_3, v_4)\}$, $\{(v_2, v_3), (v_1, v_4)\}$, $\{(v_4, v_5), (v_2, v_5)\}$, $\{(v_1, v_5), (v_3, v_5)\}$, $\{(v_2, v_4)\}$, $\{(v_3, v_1)\}$.

It is worth emphasizing that the selection of an effective characteristic function has a key influence on the number of update iterations. As an example, consider choosing a characteristic function $f(x_1, x_2)$, which takes two variables as its input, and should differentiate between, say, $(2, 6)$ and $(3, 5)$. It is obvious that $f(x_1, x_2) = x_1 + x_2$ is not a good choice, since it cannot reflect the differences between the two sets of inputs, but $f(x_1, x_2) = 2 \times x_1 + 3 \times x_2^2$ might be a better selection. Also note that as we update the characteristic matrix, additional topological information is generated between the graph entities, resulting in more topological details about the graph. Our selection process is as follows: a list of functions, whose forms suggest their potential

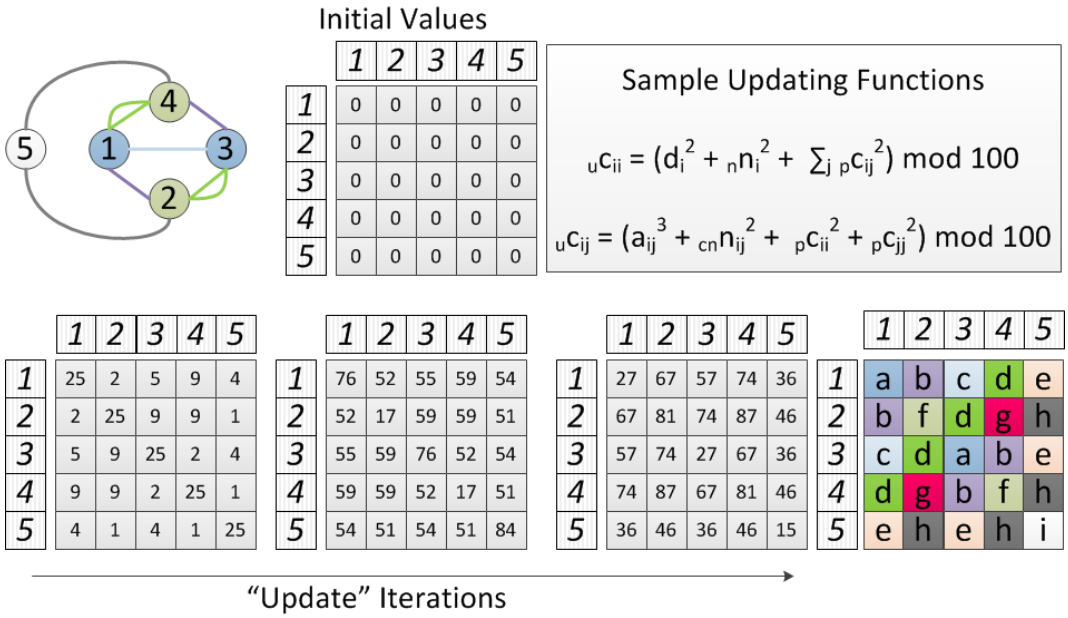


FIGURE 2.2.4: The characteristic matrix captures the similarities between vertices or vertex pairs. In the sample update functions, ${}_pc_{ij}$ and ${}_uc_{ij}$ are respectively the previous and updated values of the element of row i and column j of the characteristic matrix, ${}_nn_i$ is the number neighbors of vertex i, ${}_{cn}n_{ij}$ is the number of common neighbors between vertices i and j and a_{ij} is the element of the the adjacency matrix, reflecting the number of edges across vertices i and j.

to distinguish between different sets of inputs are compiled in a list; then tests are performed on them to measure their efficacy in finding similarities which is done by comparing their outputs with those of well-established conventional techniques [24, 52, 113] for correctness and by finding the number of required updates for each function for computational efficiency; the best performing functions are selected to be used for isomorphism detection.

The nature of the described process implies the possibility of false positive occurrence in the isomorphism test, if the updating functions are not selected carefully, meaning that two non-isomorphic topologies may be identified as isomorphic. Although we have illustrated in section2.3 that with good selection of the updating functions, not even a single such case is observed, we also propose a solution for when the functions are poorly chosen: the rows and columns of the adjacency matrices associated with each pair of topologies which have been identified isomorphic are rearranged as suggested by their corresponding characteristic matrices and then the resulting adjacency matrices are compared to test the isomorphism hypothesis. This additional step unlike the traditional shuffling method does not require exhaustive investigation of all possible rearrangements and thus is less expensive than it by orders of magnitude.

Constructing Unipartite Subgraphs As already mentioned, once the number of edges allocated to each subgraph is known (i.e. when we have the coarse adjacency matrix), the first phase in the graph enumeration stage is to generate unipartite subgraphs (i.e. specifying the elements of the square submatrices located on the diagonal of the fine adjacency matrix). We again apply the divide and con-

quer paradigm to classify subgraphs constructed on a group of vertices into distinct link families. The link family here is determined by the vertices of a given degree group, and the degrees are internal to that particular group, i.e. they reflect the number of connections for each vertex inside the vertex group. Thus, the task is to find all the valid link families for the group, and we follow the same process as the one described above. However, here vertices of the group are partitioned based on their degree within the subgraph. To exemplify, consider the case of enumerating graphs for the 6-bar linkage with ($\bar{a}_{11} = 4, \bar{a}_{12} = 4, \bar{a}_{22} = 2$) as its coarse adjacency matrix, for which the first two tasks focus on constructing unipartite subgraphs. For the first degree group, we have $|L| = 4$ and $|J| = 2$, which gives us the following families: ($|L_0| = 0, |L_1| = 4, |L_2| = 0$), ($|L_0| = 1, |L_1| = 2, |L_2| = 1$) and ($|L_0| = 2, |L_1| = 0, |L_2| = 2$). For the second degree group, we have $|L| = 2$ and $|J| = 1$, for which we can generate the following families: ($|L_0| = 1, |L_1| = 0, |L_2| = 1$) and ($|L_0| = 0, |L_1| = 2, |L_2| = 0$).

Base Cases for Unipartite Graphs The base case for the vertex partitioning into degree groups is reached when all vertices in each group have the same internal degree, when the problem becomes one of generating *regular* graphs [40]. By definition, an r-regular graph is one for which the degree of all the vertices is equal to r. The following three scenarios may arise in the process:

1. $r \leq 2$:

    (a) $r = 0$: only one graph can be generated, which consists of $|L|$ single isolated vertices;

(b) $r = 1$: can result in a valid graph only when $|L|$ is an even number. The only feasible graph would have $|L|/2$ edges with two vertices on the two ends of each;

(c) $r = 2$: the graph would be a collection of isolated circular subgraphs from different *sizes* reflecting the number of vertices involved in the cycle. Enumerating graphs for this case is equivalent to specifying the number of cycles from each size. For this, we again employ a divide-and-conquer approach: let C_i denote the set of cycles with i vertices. We set the value of $|C_i|$ for the smallest i, followed by solving the subproblem for the remaining $|C_i|$ numbers. For instance, let's say we have $|L| = 5$ and $r = 2$. Since self-loops [4] are not allowed unless we are dealing with contracted graph, we know that $2 \leq i \leq 5$. Selecting $|C_2| = 0$, we come up with $|C_3| = 0$, $|C_4| = 0$ and $|C_5| = 1$. This set of numbers represents a cycle with 5 vertices. However, if $|C_2| = 1$, we have $|C_3| = 1$, $|C_4| = 0$ and $|C_5| = 0$ which reflects two cycles with sizes 2 and 3.

2. $r > 2$ and $|L| = 2$: only one graph can be generated without self-loops, which consists of two vertices connected by r parallel edges.

3. $r > 2$ and $|L| > 2$: This situation can be addressed by decomposing the regular graph into two subgraphs: (1) one in which two vertices v_1 and v_2 have degree $r - 1$ and the remaining vertices have degree r (2) one edge between v_1 and v_2. We first enumerate all possible cases for the first subgraph, as its vertices can be partitioned into two groups. Next we add one edge to each subgraph, wherever one is missing. These cases are summarized in Fig 2.2.5.

[4] A self-loop is an edge that connects a vertex to itself [10]

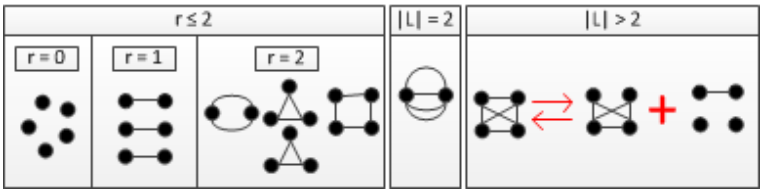


FIGURE 2.2.5: The base cases that appear in the recursive process of topology enumeration.

Enumerating Bipartite Subgraphs Following the construction of unipartite subgraphs, the next phase in the graph enumeration process is to form the bipartite graphs or, equivalently, specifying the elements of the remaining (off-diagonal) rectangular submatrices in the fine adjacency matrix. The order of specifying the elements of submatrices in the fine adjacency matrix is illustrated in Fig 2.2.6. Starting from the solutions for the submatrix associated with the subgraph constructed on the first group, at each level we add one new vertex group and accomplish two tasks: (1) choosing the elements of the diagonal square matrix from the previously determined solutions for unipartite subgraphs (2) specifying the elements of the rectangular submatrices associated with the bipartite subgraphs constructed across vertices of the new group and the vertices of groups preceding it, one at a time. The set of solutions obtained after adding each subgraph (submatrix) is tested for isomorphisms; if an isomorphism is detected it is eliminated before the next update. For the intermediate adjacency matrix solutions associated with intermediate graph solutions formed in the enumeration process, the elements of the submatrices that are not specified are set to zero, reflecting the fact that no connections are yet established in those regions of the graph.

Constructing bipartite subgraphs is equivalent with specifying the elements of the rectangular submatrices (Figure 2.2.7). Let L_i and L_j represent two vertex groups,

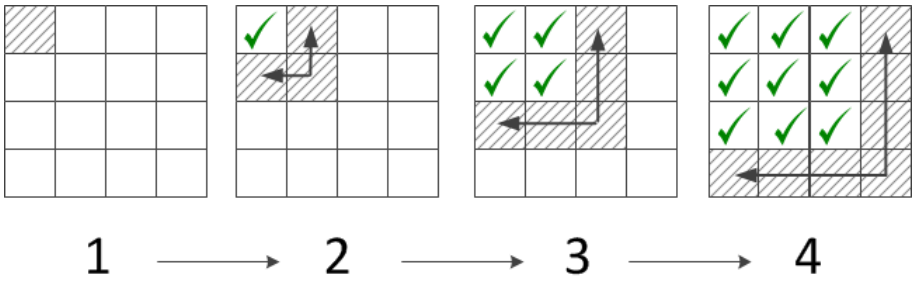


FIGURE 2.2.6: Starting from the subgraphs constructed on the first vertex group, one group is added at each step. For the new group, the unipartite graphs are constructed on the vertices of the group. As well, bipartite subgraphs are constructed between the vertices of the group and the vertices of groups preceding it, one at a time.

across which we want to form bipartite graphs. From the coarse adjacency matrix, we have the total number of edges present in the subgraph equal to $\bar{a}_{ij}$. This means that the summation of elements in the corresponding submatrix must be $\bar{a}_{ij}$. Due to the symmetry of the fine adjacency matrix, two rectangular submatrices, which are the transpose of each other, represent the subgraph. Assuming that we are considering the one in the upper triangle, the summation of elements in a row of the submatrix reflects the degree of the corresponding vertex in group i that is internal to the bipartite graph. Similarly, the summation of elements in a column of the submatrix relates to the degree of the associated vertex in group j internal to the bipartite graph. Specific criteria must be met for specifying the elements of the submatrix to represent a valid bipartite subgraph solution. The internal degree of each vertex, from either of the two groups, plus its computed degree from the previously formed subgraphs, must not exceed the prescribed overall degree of the vertex in the graph. This condition will impose upper bounds for the summation of elements in individual rows and columns of the submatrix which then will be used toward finding solutions for the rectangular submatrix. Also if j reaches its limit (i.e. when we are considering the bipartite subgraph across the last group and other groups), an additional condition must be

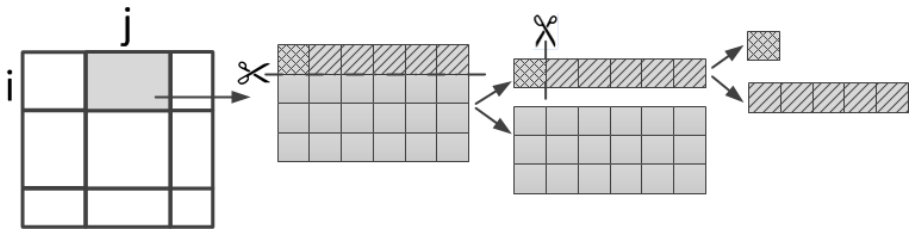


FIGURE 2.2.7: The divide-and-conquer approach for specifying the elements of the rectangular submatrix.

considered, namely the overall degree of vertices of group i must be matched.

Given the upper bounds for the summation of elements in individual rows and columns of the submatrix, the enumeration of solutions is recursively partitioned into two sub-problems: (1) define the elements of the first row (2) define a smaller submatrix, which is obtained by removing the first row from the original submatrix. The base case happens when we reach a submatrix with only one row. Solving for a row matrix is also done by taking a divide-and-conquer approach in which, at each step, a row is divided into its first element and a sub-row with one less element. The base case happens when the row has only one element, finding a solution for which is trivial (Fig. 2.2.7). Note that, when we break the problem into sub-problems, the upper bounds on the summation of elements of rows and columns are updated accordingly.

As an example, consider the case of the 6-bar linkage. We showed that several coarse adjacency matrices could be produced for such a linkage, one of which is given by ($\bar{a}_{11} = 4, \bar{a}_{12} = 4, \bar{a}_{22} = 2$). Consequently, for this case there are 2 edges internal to the group of binary vertices, 1 edge internal to the group of ternary vertices and 4 cross-edges belonging to the bipartite subgraph across the two vertex groups (Fig. 2.2.8). We showed that different internal families could be found for

the two unipartite subgraphs constructed on the two vertex groups of binary and ternary. Specifically, consider ($|L_0| = 1, |L_1| = 2, |L_2| = 1$) for the first group and ($|L_0| = 0, |L_1| = 2, |L_2| = 0$) for the second group (Fig. 2.2.8). The group of binary vertices has been divided into three subgroups, while the group of ternary vertices carries only one sub-group. In other words, for the group of ternary vertices we have reached the base case, and the two vertices of the group can be connected in a unique way (see Figure 2.2.5). However, for the group of binary vertices we continue the divide and conquer partitioning and generate coarse adjacency matrices for the sub-groups of the binary vertices group. The only possible coarse adjacency matrix is the one shown in Fig 2.2.8. The non-zero values of the matrix imply that there must be two connections between the sub-group with internal degree 2 and the sub-group with internal degree 1, as can be seen in Figure 2.2.8.

So far, the internal edges have been placed inside the vertex groups. The next step is to make 4 connections across the binary and ternary vertex groups, for which we need to define a 4×2 rectangular matrix. Each vertex has to have a number of connections with other vertices equal to its degree. Some of these come from the internal degrees, while the remaining connections will follow from those formed in the bipartite graphs. We note that the elements of the rectangular submatrix must be prescribed according to the summation criteria shown in Fig. 2.2.8. By doing so, we obtain a set of rectangular matrices, which are then combined with the rest of the adjacency matrix, and followed by isomorphism detection and elimination as outlined above. By allowing only for single edges, this process outputs one solution, namely the Watt topology of a 6-bar linkage.

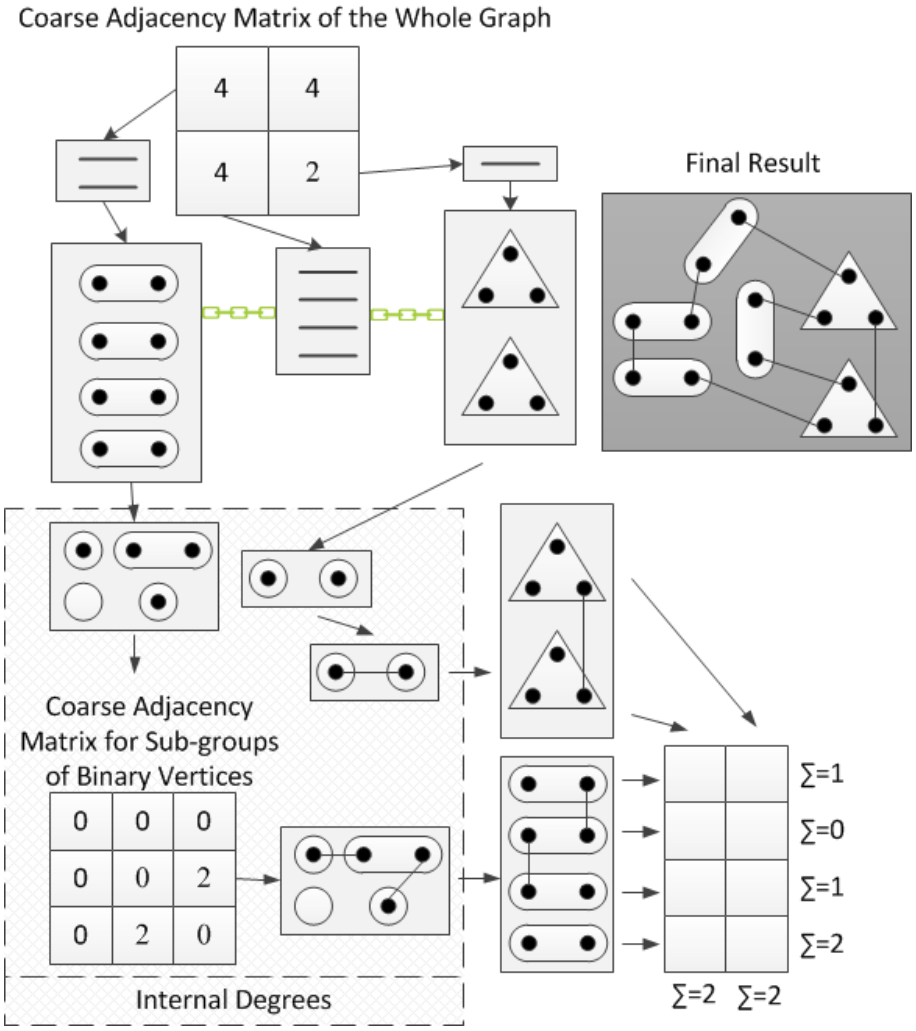


FIGURE 2.2.8: An example of a graph for a link family.

Contraction/Expansion In most practical cases, the minimum vertex degree in a topology is 2. Also binary links are more common in the enumerated families, i.e. $0 << |L_2|/|L| < 1$, in which case it is a common practice to employ the *contraction/expansion* method [166], which drastically speeds up the enumeration process. By definition, a contracted topology is obtained by replacing each binary link chain of a topology with one single edge. The contraction/expansion consists of enumerating contracted families, enumerating topologies for them and then expanding the topologies by distributing binary vertices among their edges (Fig 2.2.9). Note that, using a contraction/expansion technique requires self-loops in the contracted topologies, which would require slight modifications to the process described above.

Eliminating Topologies With Undesired *DOF* As we mentioned earlier, the Grübler-Kutzbach criterion may fail to report the true mobility of mechanism when over-constrained regions occur in the corresponding topologies. These cases can be detected and eliminated by performing an additional mobility analysis, and we use the technique presented in [146].

2.2.2 Generating Linkage Arrangements

The process described above produces valid candidate topologies. The next step is to construct, for each candidate topology, the corresponding linkage arrangements to specify: (1) which link soup component can be assigned to which vertex of the topology, and (2) what is the order in which we can pair up the connection points in adjacent links. In other words, this stage is not focused on the detailed geometry, but only on the types of links and connections between them.

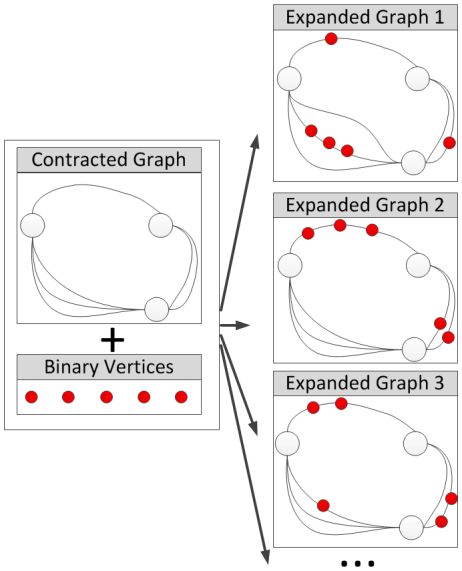


FIGURE 2.2.9: Distributing 5 binary vertices among the edges of a contracted topology.

We observe that the existence of similarities in a given topology can lead to redundancy in the linkage arrangement enumeration. For this reason, we enumerate the valid linkage arrangements in two steps: first, we enumerate all possible linkage arrangements, and then we use a slightly modified version of our heuristic method for similarity detection (the link type and the pairing information are inputs at each iteration), to identify and eliminate redundancies. Figure 2.2.10 shows one linkage arrangement for the Watt topology by using link components from a link soup that has two binary links and three ternary links.

2.2.3 Geometry

The open question that remains to complete our synthesis procedure is the following: given a linkage arrangement, can one find valid geometric solutions that would close the kinematic loops to satisfy the prescribed DOF and avoid geometric interference

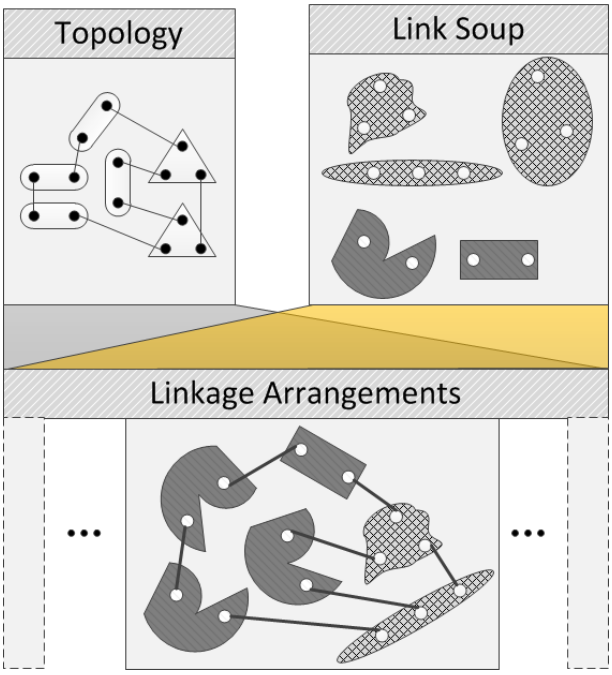


FIGURE 2.2.10: An example of a linkage arrangement for the Watt topology with links selected from a given link soup.

between the physical links? To answer this question, we first produce a reference geometry of the linkage, and then solve for the geometric constraints as described below.

Reference Geometry of the Linkage The dependencies between the joint variables that are created by the loops prevent an arbitrary selection of the joint variables. Consequently, we determine a set of *independent* joint variables by constructing a spanning tree for the corresponding topology. Starting from an arbitrary vertex as the root, we find the spanning tree of the topology using a breadth-first search protocol (Fig. 2.2.11). The collection of joint variables associated with the edges of the spanning tree describes a set of independent joint variables that can

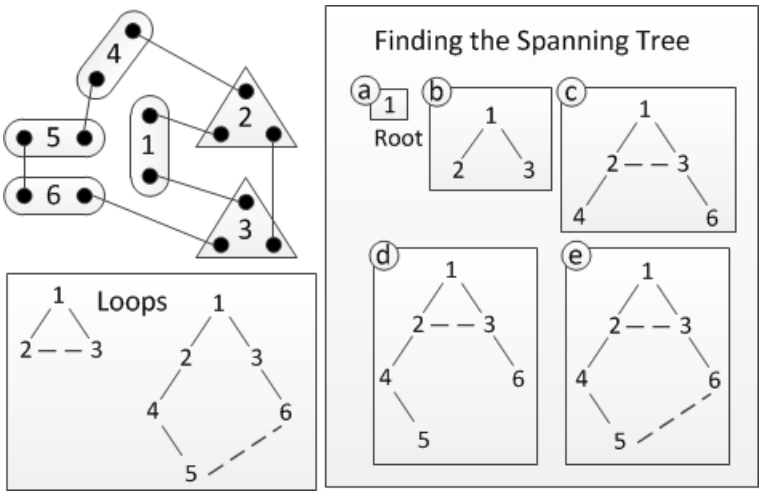


FIGURE 2.2.11: Finding a spanning tree for the Watt topology of the 6-bar linkages. Every (topological) edge that is not present in the spanning tree (shown with dotted lines) closes a loop, and is called a *chord*.

be used to fully describe the configuration of the linkage with respect to a reference geometry. In turn, the reference geometry is defined using the breadth-first search algorithm as follows: starting from a fixed geometric pose of the root link, we adjust at each level of the hierarchy the poses of the children to match the joint alignment between child and parent. This process is repeated until we find the reference poses for all links (Fig 2.2.12). The driving parameters of all joint variables (angle for revolute joints and distance for prismatic joints) are set to zero. Every edge of the (topological) graph that is not present in the spanning tree closes a loop in the graph, and these edges are called *chords*. By starting from the two ends of the chord, we traverse the tree upwards until the two paths intersect at one node, the so called loop root. The union of nodes and edges that have been visited in this process form the loop (Fig. 2.2.11).

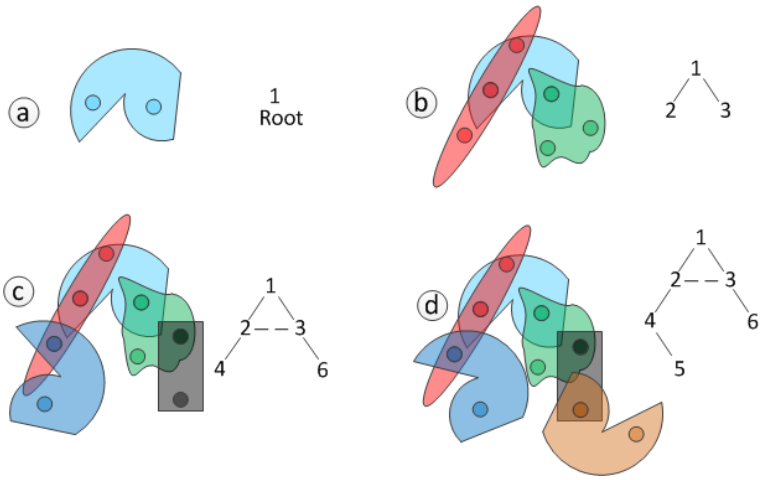


FIGURE 2.2.12: Determining the reference geometry for the linkage arrangement for the Watt topology of the 6-bar linkage.

Loop Closure Through Geometric Constraints The geometric description of the linkage now allows us to solve for the geometric constraints. The joint alignment condition has already been satisfied for the connections between children and parents in the spanning tree. This leaves us with two remaining geometric constraints: (1) joint alignment must be satisfied for the chords, which corresponds to the closure of the loops, and (2) physical clashes must be avoided.

The valid solutions are provided by those sets of joint variables for which the two constraints are simultaneously satisfied. Since an analytical solution is in general unavailable, we formulate the linkage closure as an optimization problem that aims to minimize an error measuring how 'far' a given configuration of the mechanism is from the configuration that simultaneously satisfies the constraints. Among the existing optimization techniques, we have selected a coordinate descent method (we discussed why in the optimization section) as a minimization procedure, which can be trapped in local minima. We used established random sampling methods as initial conditions for the minimization step.

Defining the Objective The objective function is defined in terms of a geometric error that approaches zero as the configuration of the mechanism approaches the ideal constraint-satisfying configuration. The threshold for an acceptable error varies depending on the selection of the link soup. For instance, for the case of molecular components, the tolerances of bond angles and bond lengths about the average values are the determining factors. Alignment of a revolute joint requires collinearity of the 4 points that define the joint axes on the two components of the joint, the axes to be in opposite directions and the corresponding 'base' points to be at a distance prescribed by the specific link soup (Fig. 2.2.13(a)). The alignment of a prismatic joint also needs the collinearity of the points defining the joint axes and the axes to be in opposite directions. In addition it requires the so called *guide unit vectors* (perpendicular to the joint axes) on the two joint components to be parallel (Fig. 2.2.13(b)). Standard distance evaluations are used to eliminate configurations that produce clashes between components, which very well fits the case of molecular components made of spherical atoms (Fig. 2.2.13(c)). The total error is defined as a weighted sum of all the individual errors.

Optimization We use a hybrid approach which, picks sample points[5] in the conformation space and then starting from each sample point, we run an optimization procedure to minimize the predefined error. Our strategy is to include only the loop closure criteria in the objective function, i.e. to ignore the effect of geometric clashes during the optimization process, followed by an investigation of the collisions in the final configuration and the corresponding range of motion. Adjusting the

[5]Any sampling method can be in principle used. However in appendix D, we will introduce a novel efficient sampling technique that is customized for the loop closure problem.

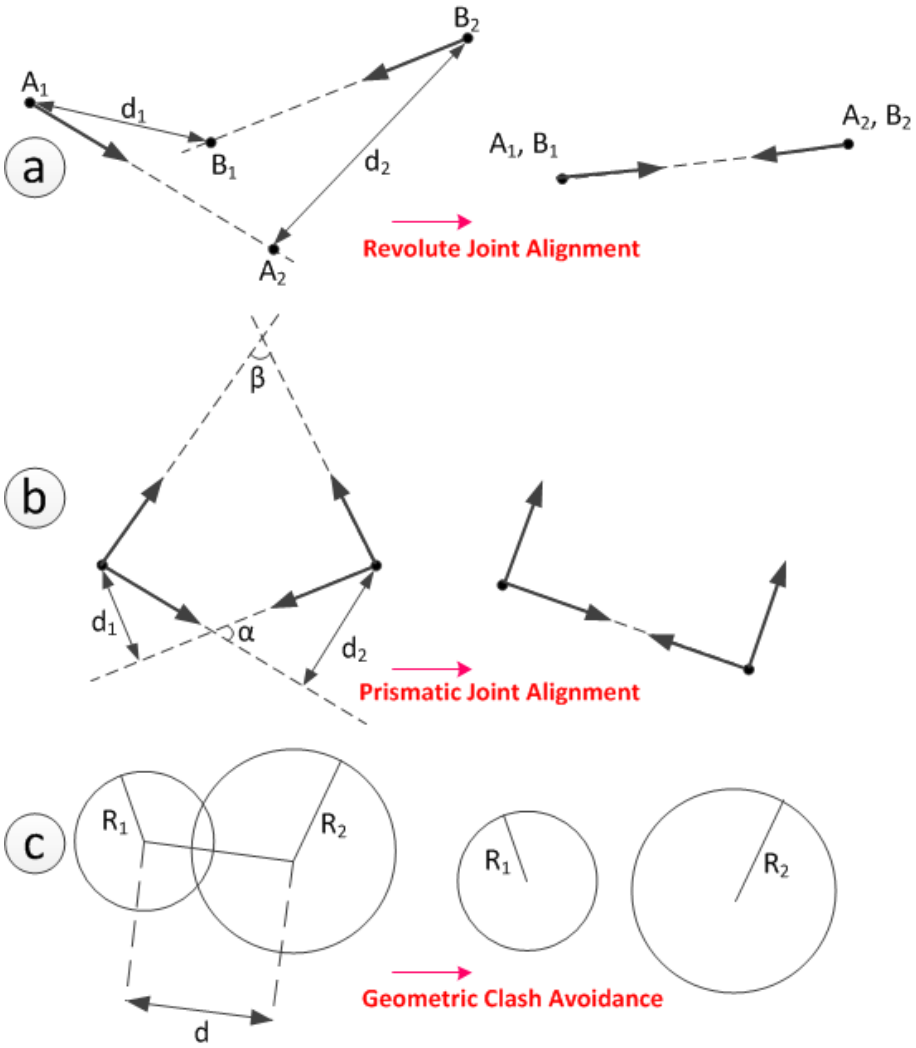


FIGURE 2.2.13: (a) For revolute joint alignment, A_1 must coincide with B_1 and A_2 must coincide with B_2. (b) For prismatic joint alignment, α, β, d_1 and d_2 must be zero. (c) For clash to be avoided between a pair of spheres, d should be greater than $R_1 + R_2$.

joint variables to close loops is one of solving the inverse kinematic (IK) problem. In [31], advantages as well as limitations of several algebraic, geometric, and iterative IK solutions have been reviewed. Specifically a comparison has been made in [176] of various aspects of Jacobian transpose method and the cyclic coordinate descent (CCD) method, including their convergence times. For our application, we have selected a CCD method. In general, CCD methods converge to solution for the majority of initial conformations, and fewer parameters must be tuned for them [120]. We iteratively lock all but one of the joint variables at one instantaneous configuration of the mechanism, and seek the optimum value for that one joint variable [27]. We then iterate through all the independent joint variables identified by the spanning tree described above, and we evaluate the closure error as the joint variables are being modified. The procedure that we use here has certain computational advantages as described below.

Two different formulations are introduced for finding the optimum value of the joint variable depending on whether the joint is revolute or prismatic. For a revolute joint, we define a local frame at the base of the joint, for which the joint variable must be adjusted. The z axis of the frame will be in the direction of the joint unit vector. We transform the problem of joint alignment into an alignment of a number of pairs of points. Let $\{(P_i, Q_i)\}$ $(1 < i < n)$ represent the coordinates of the set of point pairs whose distances must be minimized in the local frame, where P_i is the moving element and Q_i is the fixed one in space. If we denote by θ the amount of change in the local joint variable, and by P_i^0 the position of P_i associated with $\theta = 0$, we can obtain P_i at any given θ:

$$P_i = RP_i^0 \tag{2.2.5}$$

where R is the rotation matrix about the joint unit vector and can be written as:

$$\begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The closure error for any value of θ can be defined as:

$$e = \sum_{i=1}^{n} |Q_i - P_i|^2 \tag{2.2.6}$$

By substituting equation (2.2.5) into (2.2.6) we obtain

$$\begin{aligned} e = & \sum_{i=1}^{n}(|Q_i|^2 + |P_i^0|^2) \\ & - (\sum_{i=1}^{n}(2Q_{i,y}P_{i,y}^0 + 2Q_{i,x}P_{i,x}^0))cos\theta \\ & - (\sum_{i=1}^{n}(2Q_{i,y}P_{i,x}^0 - 2Q_{i,x}P_{i,y}^0))sin\theta, \end{aligned} \tag{2.2.7}$$

which can be rewritten in the following form:

$$e = a - bcos\theta - csin\theta. \tag{2.2.8}$$

By defining $sin\alpha = c/\sqrt{b^2 + c^2}$ and $cos\alpha = b/\sqrt{b^2 + c^2}$, we have

$$e = a - \sqrt{b^2 + c^2}cos(\theta - \alpha) \tag{2.2.9}$$

in which e attains its minimum value when $\theta = \alpha + 2k\pi$.

For a prismatic joint, we define a local frame whose z axis is again alongside the joint axis. If q is the amount of change in the prismatic joint variable, and P_i^0 is the position of P_i associated with $q = 0$, we have:

$$P_i = P_i^0 + \begin{bmatrix} 0 \\ 0 \\ q \end{bmatrix}.$$

The error value is defined similarly:

$$e = \sum_{i=1}^{n} |Q_i - P_i^0|^2 + \sum_{i=1}^{n} 2(Q_{i,z} - P_{i,z}^0)q + \sum_{i=1}^{n} q^2 \tag{2.2.10}$$

which can be rewritten in the following form:

$$e = aq^2 + bq + c \tag{2.2.11}$$

whose minimum occurs when $q = -b/2a$.

For those synthesized mechanisms whose loops close without geometric interference we must finally determine the range of valid motions, i.e., the range of joint variables for which the constraints on loop closure and interference free motions hold.

2.3 RESULTS

We developed several tests to investigate the viability and efficiency of the individual modules of the proposed method. We have also run our implementation on a more realistic test case to examine the practicality of the proposed method.

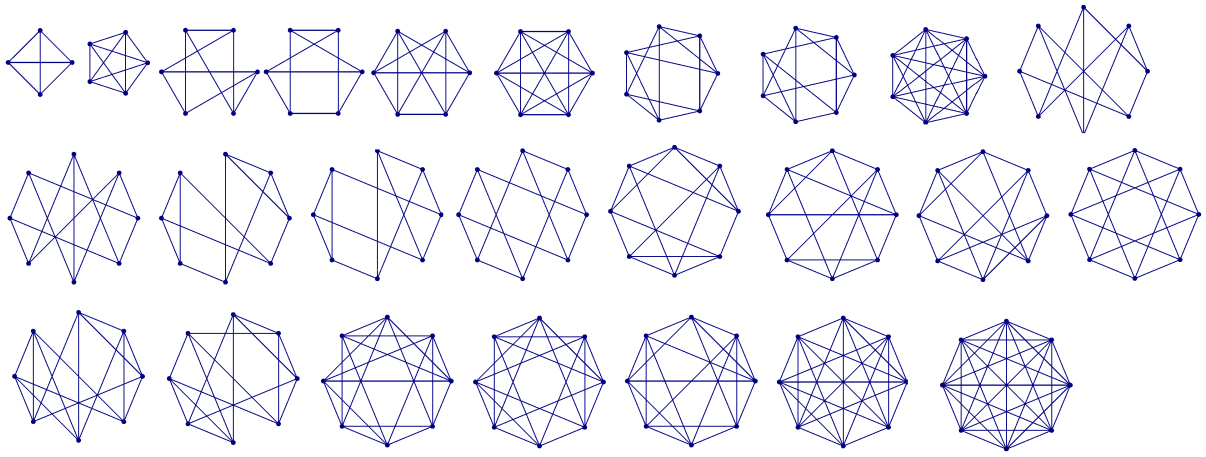


FIGURE 2.3.1: r-regular graphs with $r > 2$, with up to 8 vertices, and containing only single edges.

Testing Validity Of The Topology Enumeration Module As a benchmark for the topology enumeration technique, we generated the r-regular graphs with $r > 2$, up to 8 vertices, and containing only single edges, which are shown in Fig. 2.3.1. The results perfectly matched the results reported in [109] and show that the proposed method enumerates all such graphs.

Testing the Heuristic Similarity Identification Module The viability of the heuristic method in capturing the topological similarities and isomorphism detection was tested by generating pairs of topologies: the first topology was generated randomly; the second one could either be obtained by shuffling the rows/columns in the adjacency matrix of the first topology, resulting in a pair of isomorphic topologies, or could be produced randomly and independently of the first topology. Each pair was input to the proposed heuristic technique as well as a traditional implementation of the isomorphism detection (brute force trial of various rearrangements

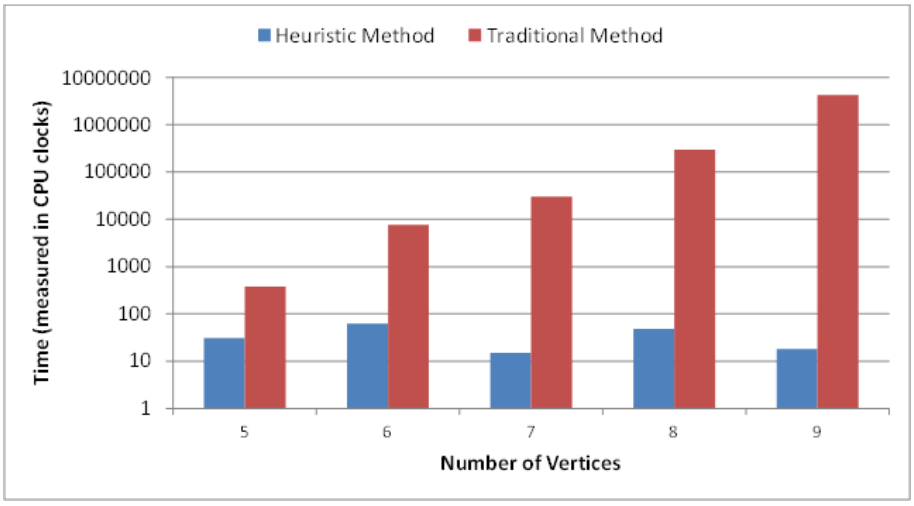


FIGURE 2.3.2: Average CPU time (measured in CPU CLOCKS) of the traditional and the proposed heuristic technique for isomorphism detection in 20 pairs of topologies.

of rows/columns [166]). The results showed that the proposed heuristic method produced the correct answer in all test cases. Furthermore, the average CPU times of the two methods for 20 pairs of topologies with increasing numbers of vertices are shown in Figure 2.3.2. The results indicate that the average CPU time of the traditional method grows exponentially as the size of the topology increases in contrast to the CPU time of the proposed heuristic method.

Validating The Optimization Module For Loop Closure To test the capabilities of the optimization module, we constructed pre-closed linkages by picking the base points and rotation axes for each joint of the linkage, which provided linkages for which loop closure was satisfied by construction. Then the loop closure was perturbed by randomly modifying the joint variables to produce an open kinematic loop. The optimization module was then used to search for the loop closure configuration of these linkages. For each linkage, a number of solutions were obtained, which

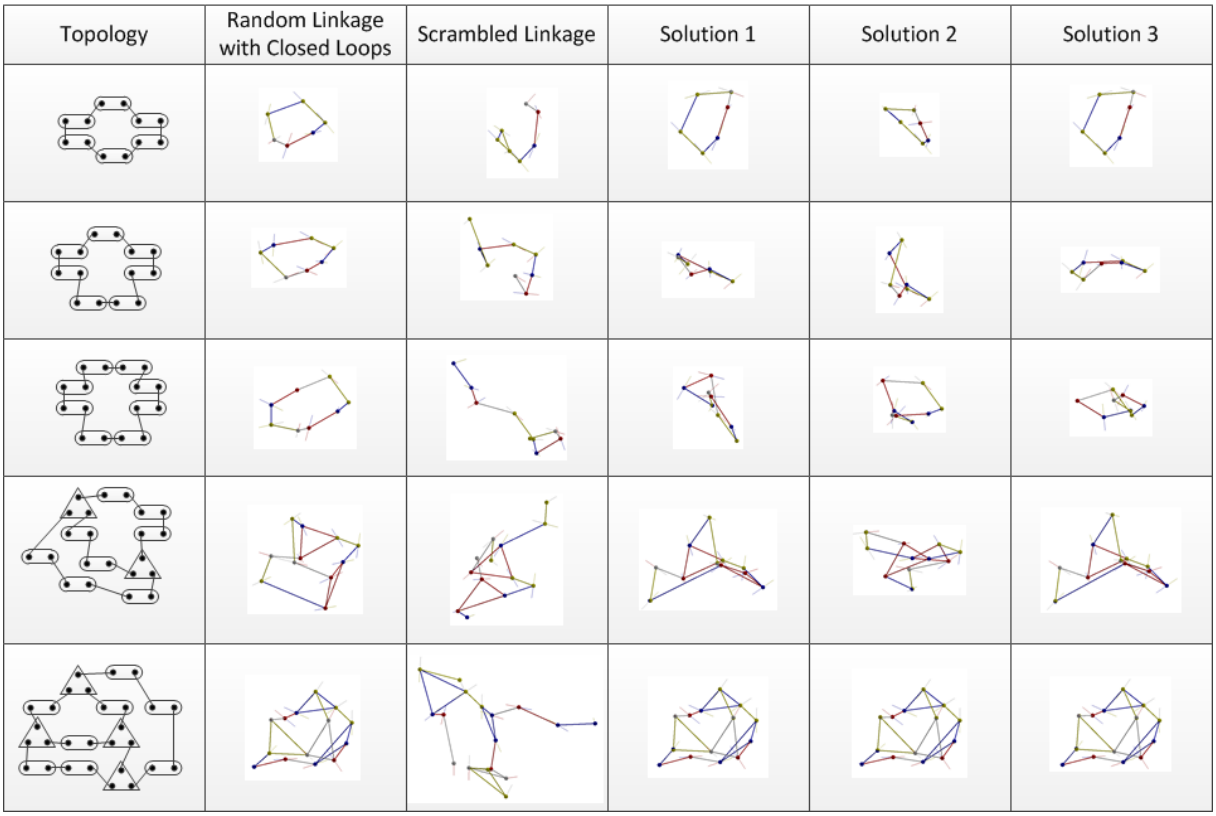


FIGURE 2.3.3: Closing open loops with known solutions.

may or may not fall into the same solution branch for the mechanism. Figure 2.3.3 demonstrates this procedure for a number of spatial topologies, and we show the first three solutions that we generated for each topology.

Note that the binary links are represented with single bars with two spheres at the ends, whereas the ternary links are represented by triangles consisting of 3 bars and spheres at the corners. The line segments protruding from the spheres serve as the joint axes. As can be seen in the figure, the closed loop configuration displays aligned joint axes, corresponding to a zero closure error. Note that even though we started from a known closure configuration, a given linkage may have different branches of solutions, or the same closure can occur for different sets of joint variables[6], both of which are reflected in the configurations displayed in Figure 2.3.3. Our tool has a module that captures the ranges of motion for these solutions, based on which, solutions that fall into the same branch are identified and redundant solutions are discarded.

Constructing 1-*DOF* Linkages From Molecular Components One of the major applications of the developed technique is in the synthesis of novel molecular machinery. The critical aspect in constructing molecular linkages with constrained motion is the feasibility of loop formation. Fortunately, several approaches to loop formation are known, such as head-to-tail cyclization [39] and disulfide bond formation [179] in proteins. Many molecules, other than proteins, can have loops such as synthetic cyclic polymers, circular DNAs, cyclic polysaccharides and Rotaxnes [141].

[6]For example, a 4-bar mechanism maintains the closure during a range of values of the joint variables.

Figure 2.3.4 depicts how the compiled link soups from the molecular components[7] are combined with topologies to yield linkage arrangements. The open-loop configurations of linkages are then set up based on these arrangements and then passed to the optimization module to get the kinematic loops closed. Finally the range of motion is computed for each linkage. The resulting collection can be found advantageous particularly in synthetic chemistry, as it narrows down the vast design space into a limited number of possibilities, with simulated behaviors for each candidate. Specifically, combining the unique capabilities of this package with the insight of the chemists/biochemists can lead to more efficient and more structured design processes in the smaller scales.

[7]We used Avogadro open source software to extract the link components [72]

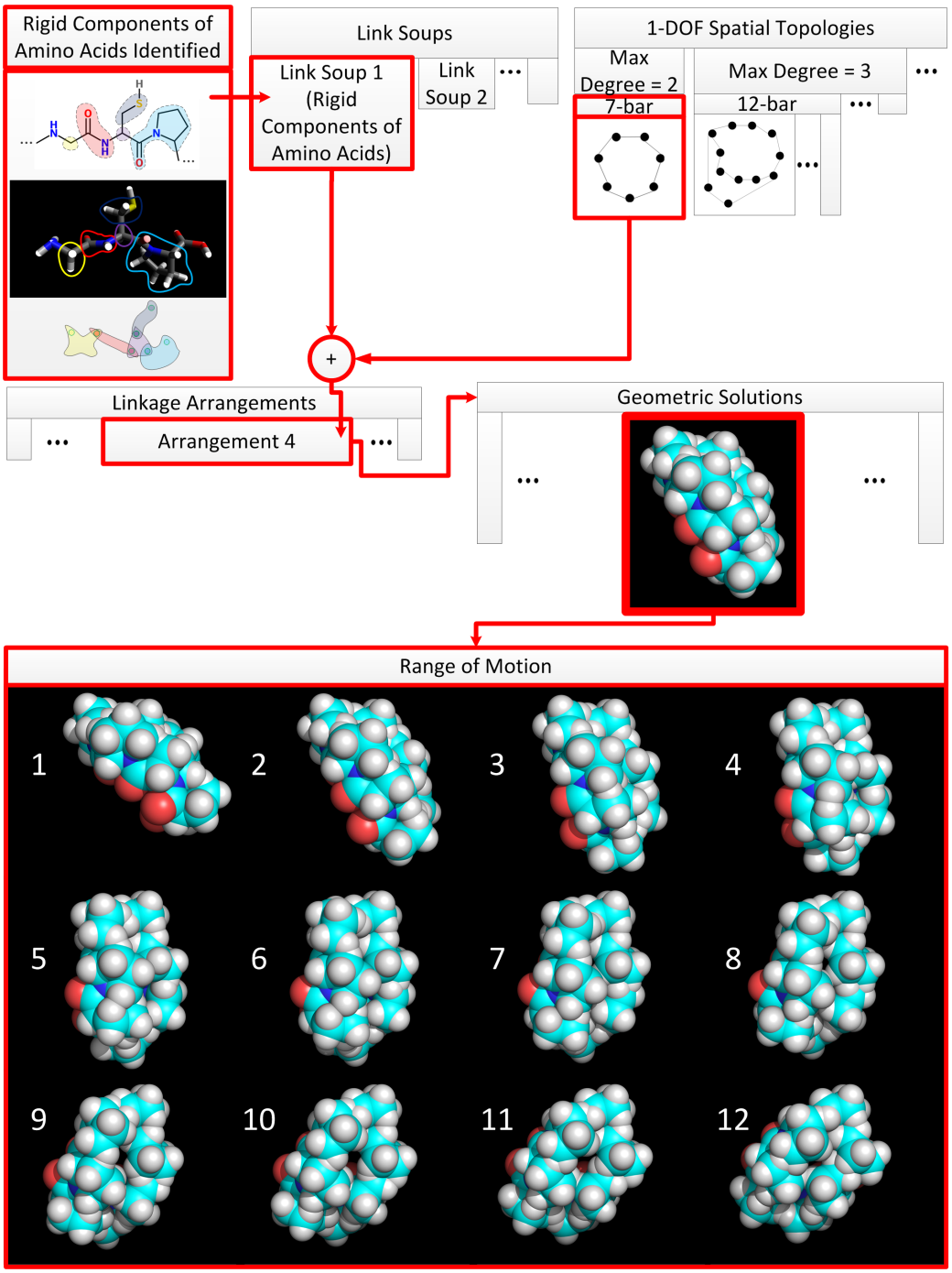


FIGURE 2.3.4: A 7-bar linkage is synthesized from rigid fragments of amino acids. Even though the linkage closes geometrically, the stability of the molecule is yet to be investigated.

Chapter 3

Systematic Design, Analysis and Control of Manufacturable Nano Machines

3.1 INTRODUCTION

Natural machinery in all scales from nano (e.g., biomolecules) to micro (e.g., cells) to macro (e.g., organs and bodies) are products of evolution. These machines are robust yet usually very complex in structure. Design methodology, although in many cases is inspired by natural products, provides a more reliable alternative for **artificial** synthesis of **novel** machines, thanks to the unique features it has: the designed machines have fewer number of components with respect to naturally existing machines and thus are simpler to build; the behaviours of the designed machines are easier to predict and thus these machine are easier to control; and yet, the designed machinery are capable of functions done by the naturally existing equivalents. This can be vividly

observed by comparing biological bird and manufactured airplane. Despite drastic advances of machine design in macro scale, unfortunately design methodology at the molecular scale is still very much at its infancy. Through the long process of evolution, nature has come up with optimized genetic data that provide reliable "building instructions" to cell bimolecular factories, *ribosomes*, to from long chain molecules out of prevalent naturally existing building blocks, namely only 20 different types of amino acids. These chains consequently fold into 3-dimensional functional structures [36]; the functions of many of them are derived from their specific controlled motion properties [23] and we refer to them as "molecular machinery". The fact that there are theoretically infinitely many combinations of these 20 building blocks that have not be tried by nature and that there are so many other molecular components that can be employed as building entities, suggests a huge potential for artificial fabrication of nano structures with controllable motions. The ability to manipulate motion at the nano-scale can provoke downstream physical phenomena at different scales. The immediate medical application would be controlling the motion of molecular machines to fabricate programmable nano-robots which can act as smart drugs to perform critical activities in diagnosis and curing diseases as well as empowering the immune system of living bodies because of their additional capabilities with respect to immobile inhibitory drugs. Also, combining the motions of individual machines in an ensemble can take the material synthesis to another level [152], where the designer can achieve the optimal material characteristics by making adjustments in the nano scale. As a result, artificial fabrication of novel molecular machines has become a very hot topic in the past few decades. Two major existing classes of artificial molecular machinery fabrication attempts are: reverse engineering the folding process [73] and taking the macro-scale design approach. In the former which mimics

natural processes, the designer tries to project the product of folding for a number of "instruction" sets, followed by encoding promising genetic data through ribosome to get the desired structures. Lacking a good understanding of folding mechanics has afflicted the success rate of this method, making it expensive and time-consuming, specially when a structure with controllable motion must be achieved. The current implementation of the latter approach also faces important challenges. Since at the molecular scale, building block components from arbitrary shapes and sizes are not available and different physics are governing, machine design techniques in their conventional forms are not quite practical, which leaves the designer with no option but to use their intuition and extensive trial and error.

We propose to tailor the simple, efficient and reliable techniques of macro-scale machine design to be used in nano scale towards establishing a systematic molecular design strategy. Fortunately, analogies have been identified between nano- and macro- machines that back up this idea. For instance, the classical machine components such as kinematic links and joints can be found at the molecular level as well [159]. It must be however noted that, despite the existing similarities between design at the two scales, there exists one big difference which places difficulty on the way of using conventional mechanism design methods: in the macro-world, the designer can freely choose the shapes and sizes of the machine building blocks; but unlike that, at the molecular scale, the designer is limited to the naturally existing constructional entities or the ones that can be artificially synthesized. We propose that, to address this limitation, various combinations at which the existing components can be put together must be tried until design solutions with desired motion and function properties are reached, similar to building structures out of Lego pieces. We aim at designing machines that are irreducibly simple. This simplicity must be manifested

both in (a) structure, i.e. number of used components and the connections between them, for an easier manufacturing process, and (b) mobility, i.e. number of degrees of freedom (DOF) or in other words the number of independent variables needed to fully describe the motion of a machine to give rise to controllable motion. In fact we will stick with the traditional tendency towards $1DOF$ machines which produce predictable and repeatable motions. The task of "design for function" is proposed to be broken into three smaller tasks: (1) design for mobility out of a library of building block components to reach at machines with constrained and consequently controllable motions (preferably $1DOF$) (2) devising control mechanisms to manipulate the resulting motions and (3) functionalizing the fabricated machines, to be exploited in different applications individually or as parts of an ensemble. A computational framework is developed to carry out a good portion of the aforementioned items as well as additional tasks such as compiling suitable input libraries and conducting physicals simulations. Moreover, the outputs are validated via various physical experiments. Figure 3.1.1 shows the schematics of the whole process. Our preliminary physical tests support the formation of a one-degree of freedom cyclic pentapeptide as speculated by our simulations. Furthermore, highly promising design cases can be collected in a nano-machinery ATLAS which acts as a powerful resource for synthetic chemists, biochemists, biologists and pharmacists and narrows down the vast and complex design space of molecular design.

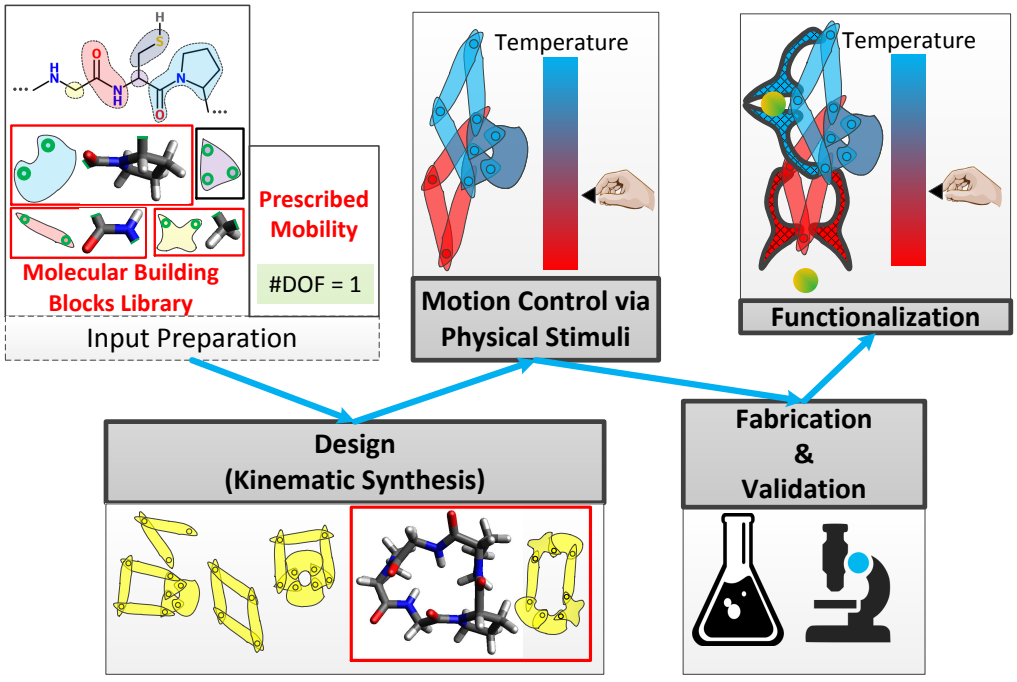


FIGURE 3.1.1: From building block components to controllable functional machines.

3.2 METHODS

Motivated by the observed resemblance between existing molecular machinery and the classical mechanisms, we want to apply the well-established knowledge of mechanism design in a custom fashion such that they accommodate design and control in nano scale. Conventional mechanism synthesis consists of three steps: (1) *type synthesis* in which the type of the mechanism is selected (2) *number synthesis* in which the topology of the mechanism is specified and (3) *dimensional synthesis* in which sizes and shapes of constructional components are determined. The analogy between single covalent bond and kinematic revolute joint as well as existence of rigid fragments in biomolecules hints that perhaps linkages-type mechanisms best reflect the molecular machinery.

3.2.1 Input Preparation

Because of scale-induced limitations, tailoring nano components to arbitrary shapes and sizes is not an option and instead, different combinations of the available components must be tried for reaching at a solution. A prosperous design process demands attentive compilation of the input building blocks as well as proper selection of synthesis criteria (e.g., which two components can be paired and how?). Building blocks must be naturally prevalent or can be synthesized with minimal effort; they must be capable of forming useful connections with each other; putting them in assembly must be achievable with current existing chemical synthesis techniques; and most importantly, they must satisfy the very basic assumption of the synthesis problem at hand, which is that, we should be able to take them as **rigid** links. In fact, the accepted extent of flexibility is determined by the designer based on the application they have

in mind. Natural and synthesized polymer molecules are reliable resources both for extracting building blocks and setting the rules of synthesis. For instance, formation of protein chain and the disulfide cross-links suggest a set of building blocks with one, two or three connection spots. In order to save time and effort during the post-design validation stage, we developed a systematic approach for meticulous selection of inputs out of existing molecular databases (Figure 3.2.1).

3.2.2 Design (Kinematic Synthesis)

In [161], we introduced a novel approach for synthesizing functional mechanisms for a prescribed DOF (preferably set to 1 as discussed in the Introduction) and made from an existing library of building block components referred to as the link soup. This approach particularly aims at considering all the possibilities and not missing one to compensate for the restraints both in terms of the limited link soup and the non-manufacturability of some candidate solutions. Therefore, different solutions are enumerated through a hierarchial process (Fig 3.2.2). Letting L and J represent the sets of links and joints in a synthesized mechanism, first, based on the input DOF, several ($|L|$,$|J|$) combinations that satisfy Grubler Kutzbach mobility criterion are enumerated, where $|.|$ is the cardinality operator. Then for each

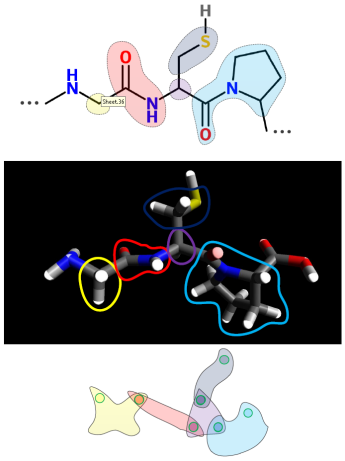


FIGURE 3.2.1: Compiling building block library.

pair, link families are generated, where a link family indicates how many links from each type are present in the mechanism. For each link family, non-isomorphic topologies are enumerated using a divide-and-conquer technique. Next, links from the link soup are allocated to the vertices of each topology in all non-redundant ways to generate a set of linkage arrangements. Note that a linkage arrangement only carries information about the assigned link components and the connections among them and **not** their spatial positioning and orientation. Finally different geometric embeddings are produced, through positional analysis (which replaces dimensional synthesis step of conventional synthesis), for every linkage arrangement. These geometric solutions will meet two geometric criteria, namely closure of kinematic loops to guarantee the predicted constrained motion and the prescribed mobility (e.g. $\#DOF = 1$), and non-overlapping links in order for the solution to be physically plausible.

3.2.3 Control Strategies

To exploit the constrained motion yielded from the specific assemblage(s) of the building blocks, appropriate control mechanisms must be devised. Scale-induced differences between macro- and nano-control strategies appear in actuation and sensing as well as form and timescale of transition between states. Instead of macro-actuation mechanisms, such as motor force or torque, and electromechanical sensors, motion at the nano scale must be manipulated and controlled by molecular-level regulation factors such as temperature, voltage, pH, light, ATP and electrochemical gradient. Furthermore, unlike conventional macro systems, where state variables often experience a continuous range, in molecular systems which are usually viewed as ensembles, a finite number of energy minima represent the feasible states (as dictated by Boltz-

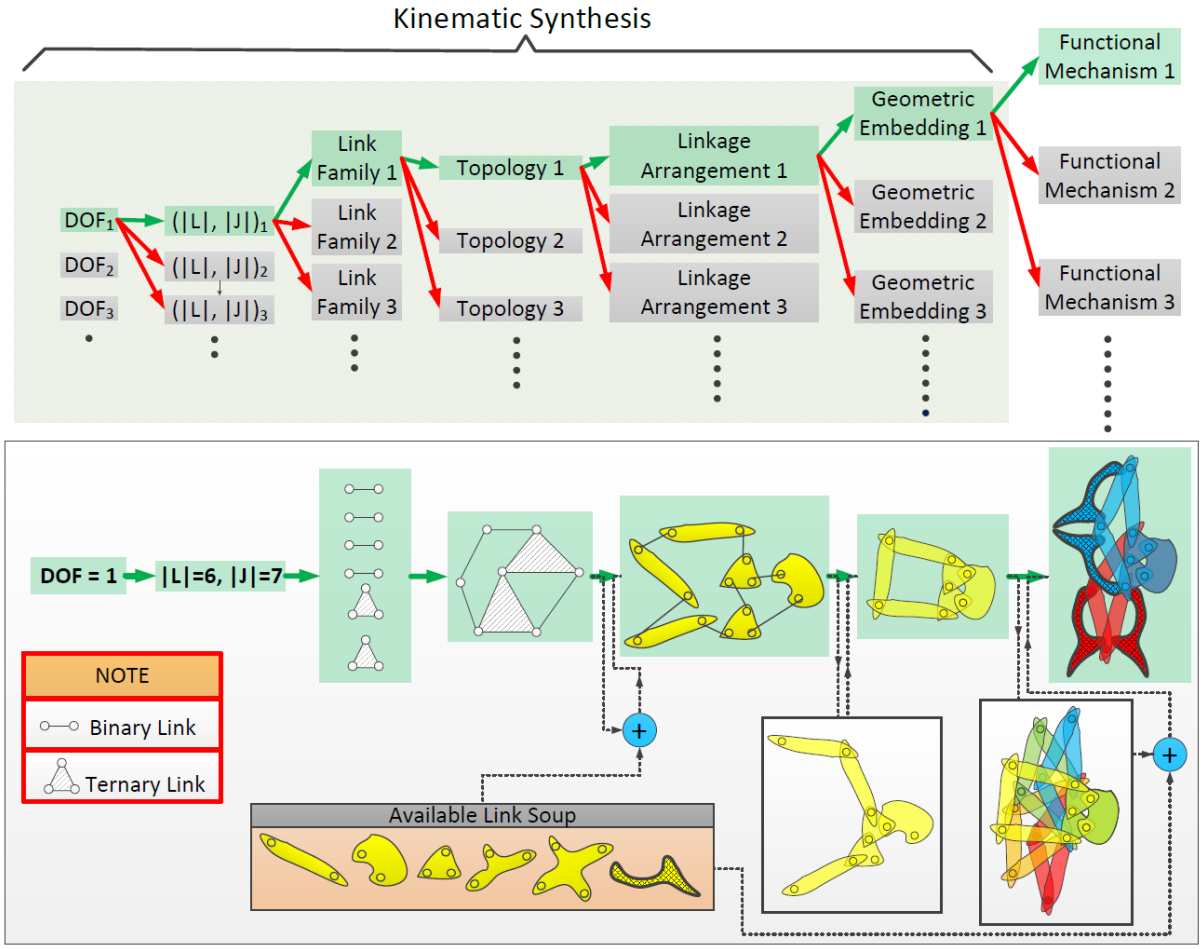


FIGURE 3.2.2: Kinematic synthesis and functionalization.

man Distribution [97]). Stable molecular states are confined with reasonably large energy barriers.[1] In order to stimulate a molecular machine to go from one conformation to another, the morphology of the energy profile must somehow change to give rise to displacement of global minimum (theoretically associated with the dominant conformational structure) in the energy profile along the axis (axes) associated with the conformational change of the molecular machine, and consequently to give motion to it, through the occurrence of one or both of the following scenarios:

1. the location of local energy minimum that, is associated with the global minimum, is shifted (**b** to **a** in Fig. 3.2.3);

2. global minimum, originally taken by one local minimum is taken over by another local minimum (**b** to **c** in Fig. 3.2.3).

In order to achieve an effective control strategy, our computational framework enables evaluation of the energy profile under the effect of different environmental conditions. Environmental parameters must be adjusted to find the optimum variation scenario for achieving the desired motion and function.

3.2.4 Fabrication and Validation

Once designed and tested in simulation, candidate mechanisms must be chemically synthesized. The main challenge here is forming molecular loops [141] to ensure the predicted constrained mobility [166]. Although, everyday advances in the synthetic chemistry open new ways to design more complex molecules, we are still too far from

[1] Otherwise, due to rapid thermal fluctuations, present at the molecular scale, only an average effect of multiple states is observable.

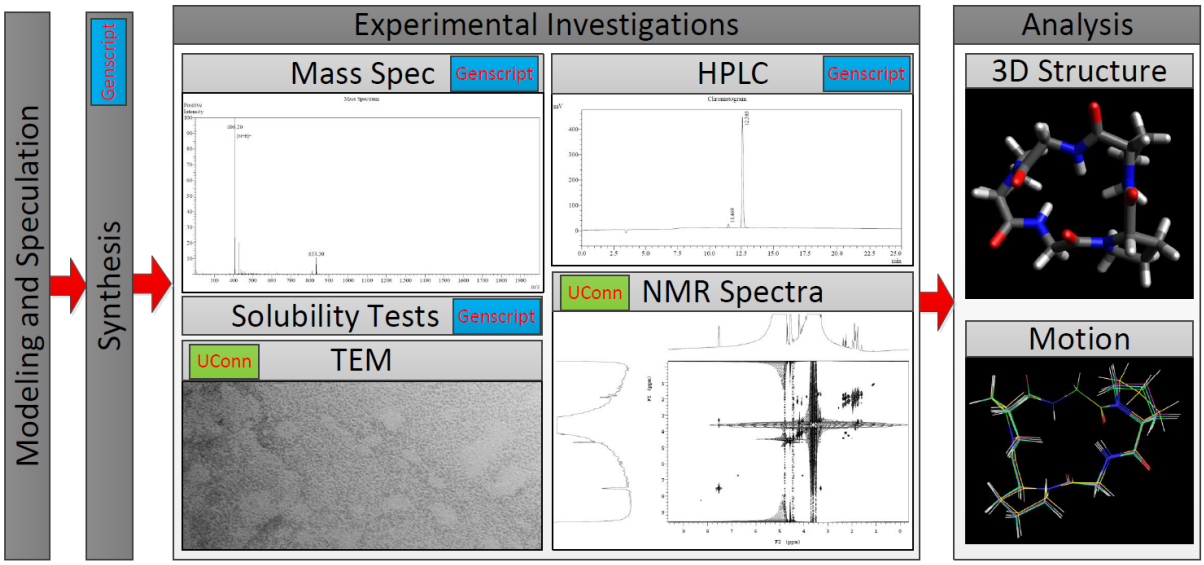


FIGURE 3.2.4: Experimental Investigations.

the point that we can manufacture any arbitrary structure. More so, not every fabricated molecule may show the speculated behaviour. Therefore it is highly necessary that we provide the synthetic chemist with multiple candidate structures (all for one same purpose) and let them choose the most promising one(s) based on their insight and experience. Once fabricated, several tests must be conducted on the resulting chemical compound to validate the predicted structure and mobility. These tests range from Mass Spectrometry (MS) to Transmission Electron Microscopy (TEM) and Nuclear Magnetic Resonance (NMR) (Figure 3.2.4).

3.2.5 Functionalization

In the final stage of the process, we seek to answer the following question: can we enhance the fabricated machines to make them functional either as individual machines or as parts of an ensemble. We put our focus on motion-induced functions.

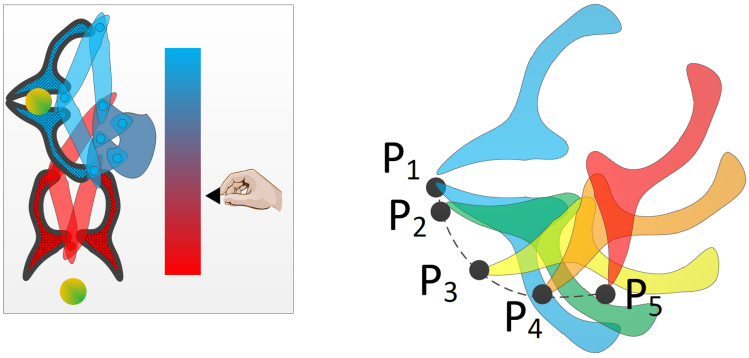


FIGURE 3.2.5: Functionalization

Therefore a critical aspect of our design process would be the quantitative representation of molecular function as some controllable motion. For example, to arrive at a melanism that can grip matter, one may make use of two levers (to be attached to the core linkage) each represented by an end point and generate a relative motion similar to that of a tweezer. Relative motion generation will be conducted by fixating one end point and forcing the other to pass through a finite number of precision points (Fig 3.2.5). Different nature of molecular physics makes this task more challenging than relating function and motion in macro scale. As a result, one must scrutinize the naturally existing molecular machines to find solid patterns between motion and function at the nano level.

3.3 RESULTS AND DISCUSSION

Protein molecules are capable of a wide range of functions, yet made of only 20 different types of amino acids. Therefore, it seems reasonable that we use the rigid components of these amino acids at our first attempt for systematic design of $1DOF$ nano-machines. As a result, we will employ the building block components shown in

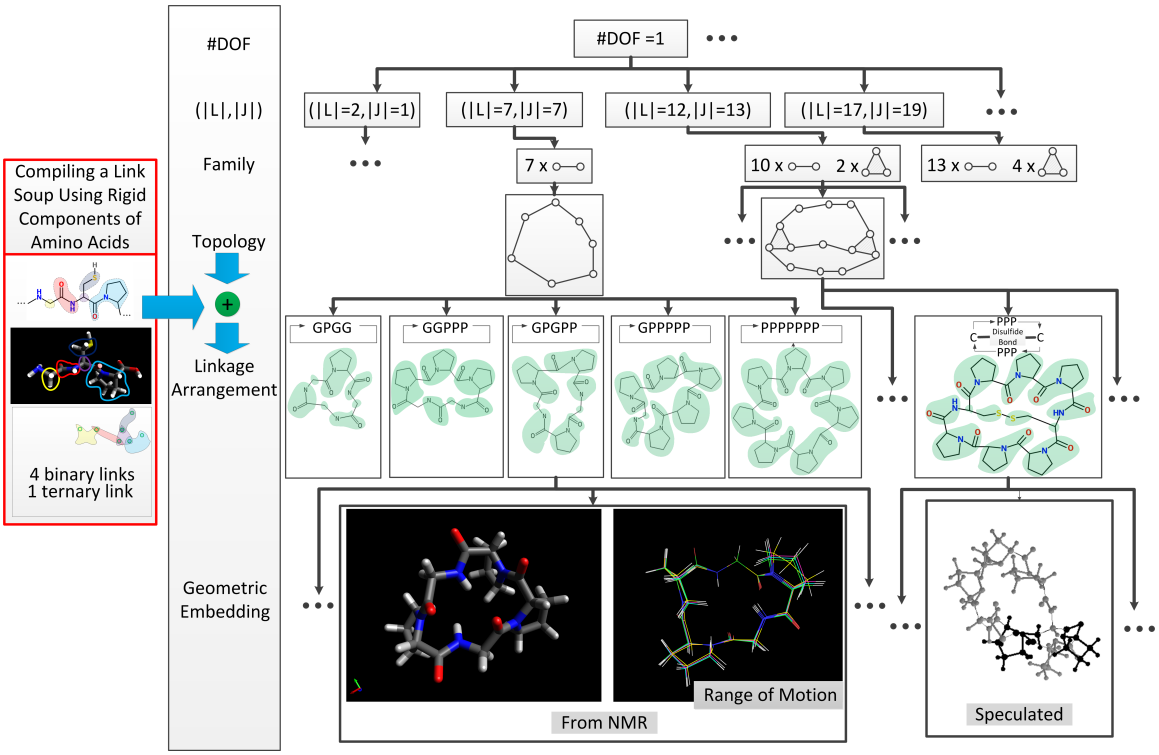


FIGURE 3.3.1: Synthesis hierarchy

Figure 3.2.1 to make functional mechanisms. We want to see what are the $1DOF$ machines that we can make with these components, given an upper bound for the number of links present in every linkage. Figure 3.3.1 demonstrates the synthesis hierarchy. Solutions for a prescribed mobility are found at different topologies and with different numbers of links. Note that, theoretically, there is no upper bound on the number of links and the limitation is enforced by level of advancement in synthesis biochemistry.

Among all feasible spatial $1DOF$ mechanisms, after the trivial case of two rigid bodies hinged together with a revolute joint, the simplest topology is a closed loop 7-bar linkage. By only using the link components shown in Fig. 3.2.1, 5 linkage arrangements for this topology can be thought of. Among these 5, we found geometric

solution for 4. 2 of these linkages were chemically synthesized by Genscript. Experimental validation were conducted (at Genscript and UConn) on the synthesized compounds to answer two questions: (1) has the chemical synthesis been successful, meaning that, does the resulting chemical compound have the expected structural formulae? (2) do we observe the predicted motion, meaning that, can the synthesized chemical be deemed as a one degree of freedom mechanism?

Figure 3.3.2 depicts MS results (conducted by Genscript) for these two compounds. The dominant mass peaks observed in the MS results, are in match with the theoretical molecular weights deemed for the two molecules, supporting that both of the compounds have been chemically synthesized and the purity of the products are reasonable.

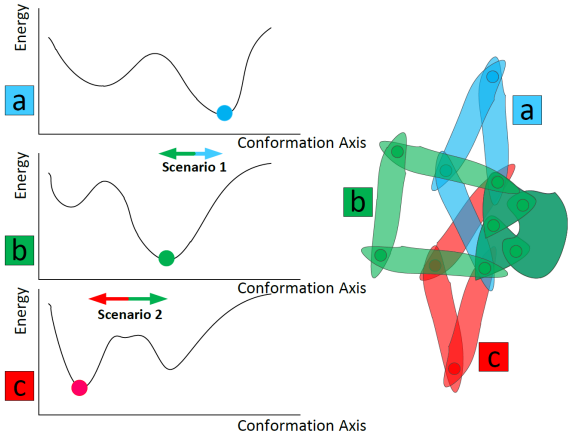


FIGURE 3.2.3: Control Strategies

As we expect conformational changes from our molecules, perhaps the best way for studying their structure and motion would be through NMR Investigations.

NMR investigations do not directly produce 3-D images. Instead, computer calculations are conducted on raw NMR data to obtain 3-D molecular models. Data is collected by placing the sample in a magnetic field, followed by emitting radio frequency signals through the sample and measuring how much of the emitted signals is absorbed by atom nuclei in the sample. Two phenomena, namely, different absorption by distinct nuclei, and the perturbing effect of adjacent nuclei on each other's

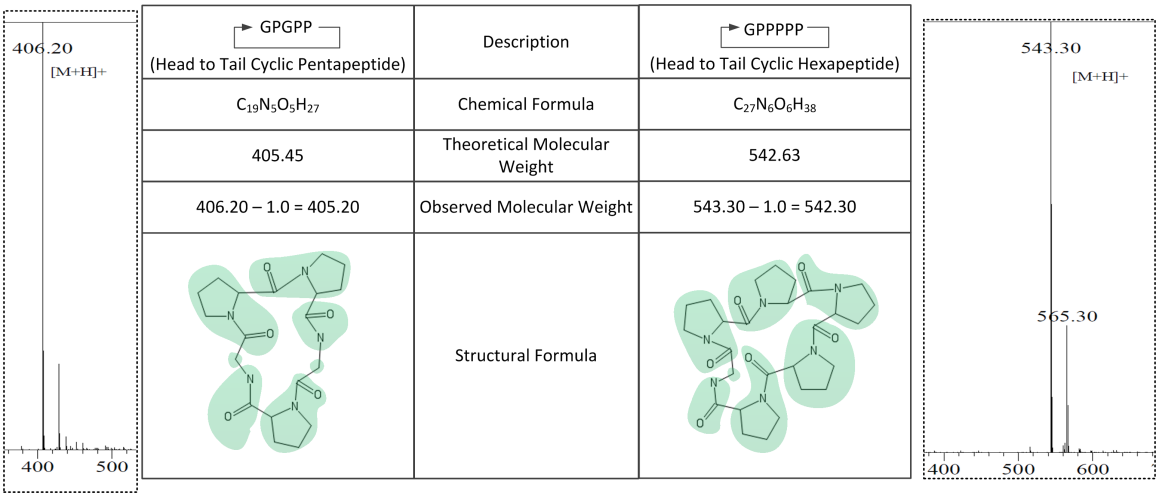


FIGURE 3.3.2: Mass Spectrometry

absorptions, give rise to one- and two-dimensional NMR spectra, respectively. The former reflects the chemical shifts of individual nuclei, by which the corresponding atoms can be recognized inside a molecule. The latter, on the other hand, yields the correlations between the frequencies of distinct nuclei, which are due to transfer of magnetization through chemical bond as well as through space. The one that is through space can be exploited for pairwise distance determinations, which in turn may be used for attaining the 3-D structure of the molecule. Solving the structure has a standard procedure. First resonance assignment is conducted to figure out which chemical shift corresponds to which atom. Then, a number of restraints (e.g., distance and angle) are generated to enable structure calculation. One way that this can work for distance is that the spatial proximity is related to NOESY crosspeaks based on the fact that the intensity of the peak is approximately proportional to one over distance to the 6th power ($1/d^6$). Because of the approximate nature of the relationship between intensity and distance, a range is given, which serves as a distance restraint. There are also a few common approaches for obtaining angle restraints

and orientation restraints. Based on the resulting restraints and other general information about the structure, it is then tried to satisfy as many of the restraints as possible, through a structure calculation process. This happens through converting the restraints into energy terms and trying to minimize the energy. The result would be an ensemble of 3-D structures [178]. Note that, representing a molecules as one single structure (as usually practiced in X-ray diffraction method) may underestimate the intrinsic flexibility of the molecule inside solution and thus a set of conformations (such as what is given by NMR) may be a better representation of experimental data for flexible molecules [8].

Through a standard procedure, one of the chemical compounds (GPGPP) delivered to us by Genscript was dissolved in DMSO solvent and various NMR experiment were conducted on the solution by Dr. Vitaliy Gorbatyuk at NMR facilities of Uconn. The NMR raw data was fed to the automated NOE assignment and NMR structure calculation package, ARIA [98] and the 20 structures with the lowest energies were selected as the candidate structures (Fig 3.3.3).

Overlapping the structures (Fig. 3.3.3), differences can be observed between them, which can be due one or both of the following two reasons: (1) alteration of joint variables along a one-degree-of-freedom motion (2) changes in shapes and sizes of the constructional links. In fact, since one of our basic assumptions has been that, the used building blocks can be taken as rigid links, we want to investigate whether shape and size changes of links can be neglected relative to alteration of joint variables or not. Figure 3.3.4 shows the ranges of changes in the 15 dihedral angles of the synthesized cyclic pentapeptide based on calculations performed on the 20 structures yielded from NMR investigations. The 8 angles that were assumed to be locked (as a result of link rigidity assumption) experience meaningfully less change than the 7 angles that are

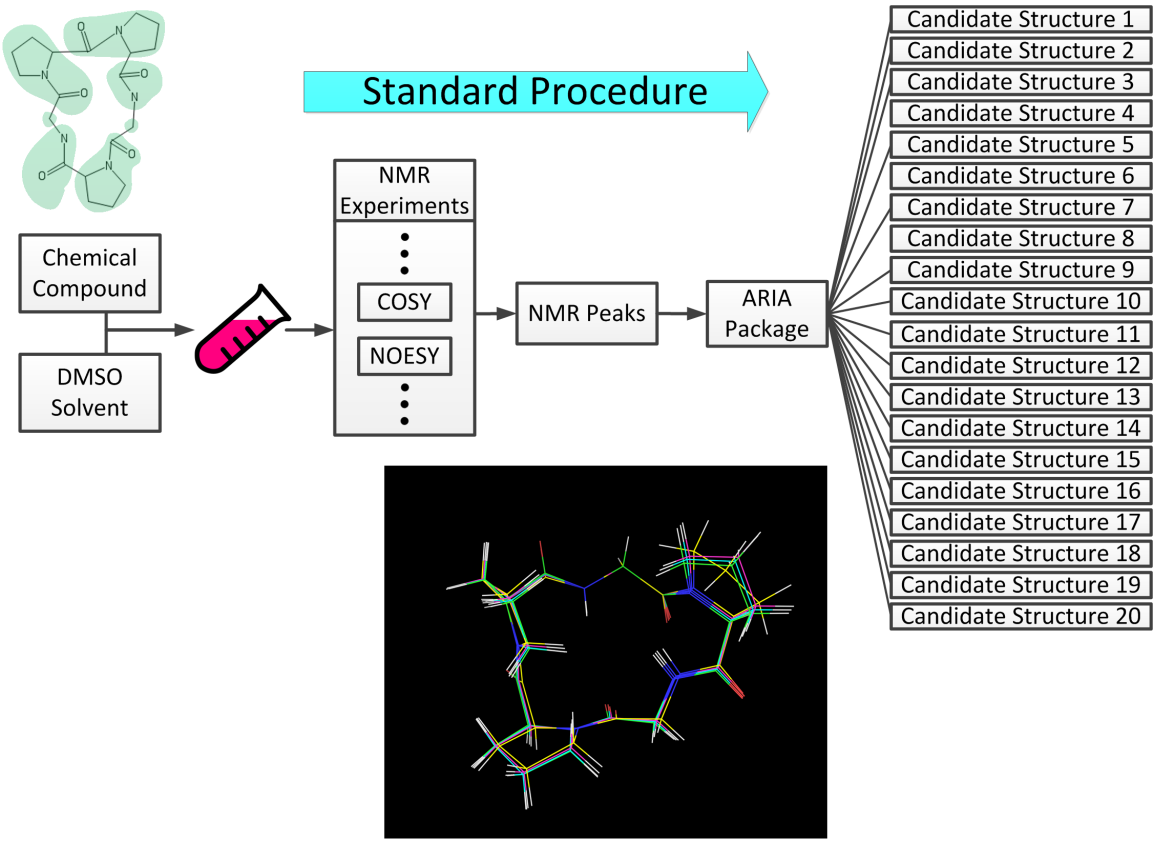


FIGURE 3.3.3: Structure calculation using NMR.

allowed to be variable in the model, even in the absence of any stimulus that causes conformational change in the cyclic molecule.

Furthermore, we take one of the 20 candidate structures (arbitrarily chosen to be number 18) and assuming that the building block components remain rigid, we speculate the kinematic range of motion (i.e. one through which the loop remains geometrically closed) and we plot the changes of the 7 to-be-variable angles with respect to one of them (referred to as the control angle), followed by superposing the 20 sets of "7 dihedral angles" calculated from NMR data on the resulting plot (Fig. 3.3.5). There seems to be reasonable match between the curves from the predicted motion and the dots obtained from the NMR candidate structures, except for two sets of 5 points that seem to be meaningfully off the curves (as depicted in Fig. 3.3.5) which must be studied further.

Investigating the structures associated with these off-the-curve dots, we find out that the configuration of the ring structure of one of the prolines is different at this group of candidate structures from the rest of the candidate structures (Fig. 3.3.6), meaning that one of the structural links is slightly different in the two groups. Therefore, we might represent the candidate structure as a model with two mobile linkages and therefore two sets of motion curves (Fig. 3.3.7).

To further investigate the on-degree-of-freedom motion model that we have deemed for our cyclic peptide and to rule out the scenario that only one single solid structure exists in the solution, we perform energy analysis. We must see whether or not all of the 20 conformations can exist in the solution. Using the software Avogadro [72] the energies of the 20 candidate structures are calculated in 4 different force fields: GAFF [173], Ghemical, MMFF94 [70] and UFF [129] (Fig. 3.3.8).

Next, using the Boltzmann distribution formula $f = exp(-E/(kT))$ (T and k

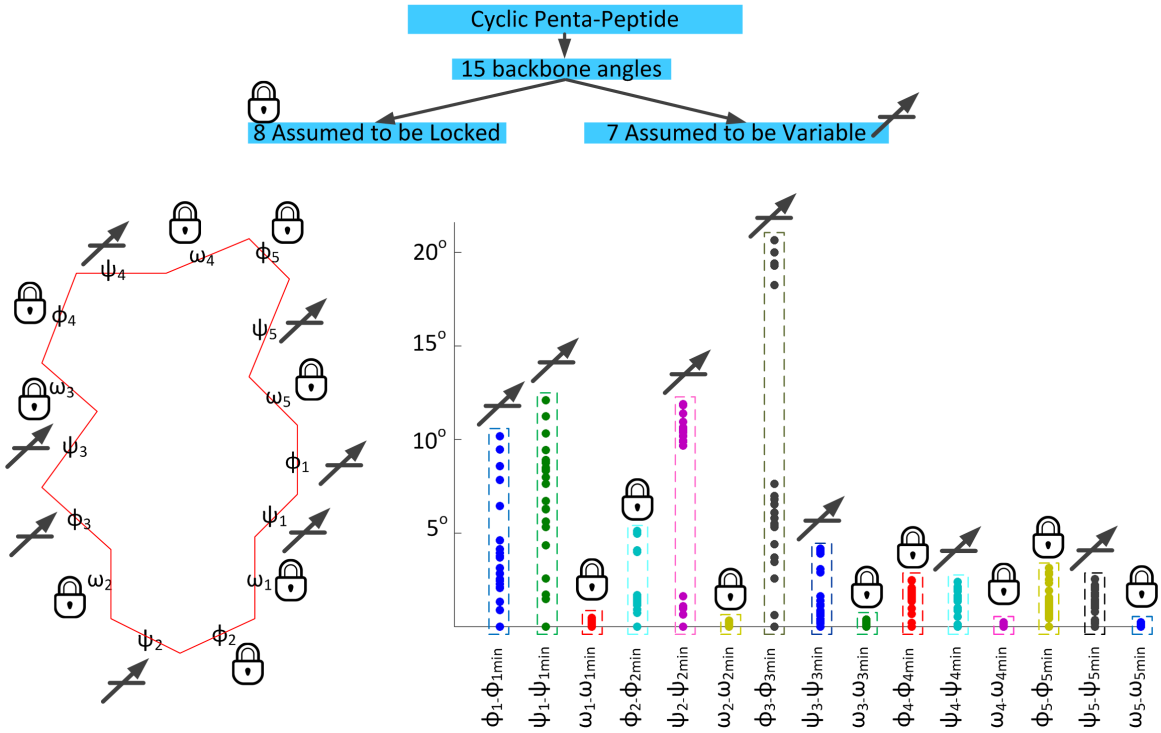


FIGURE 3.3.4: Variations in dihedral angles as calculated from NMR candidate structures.

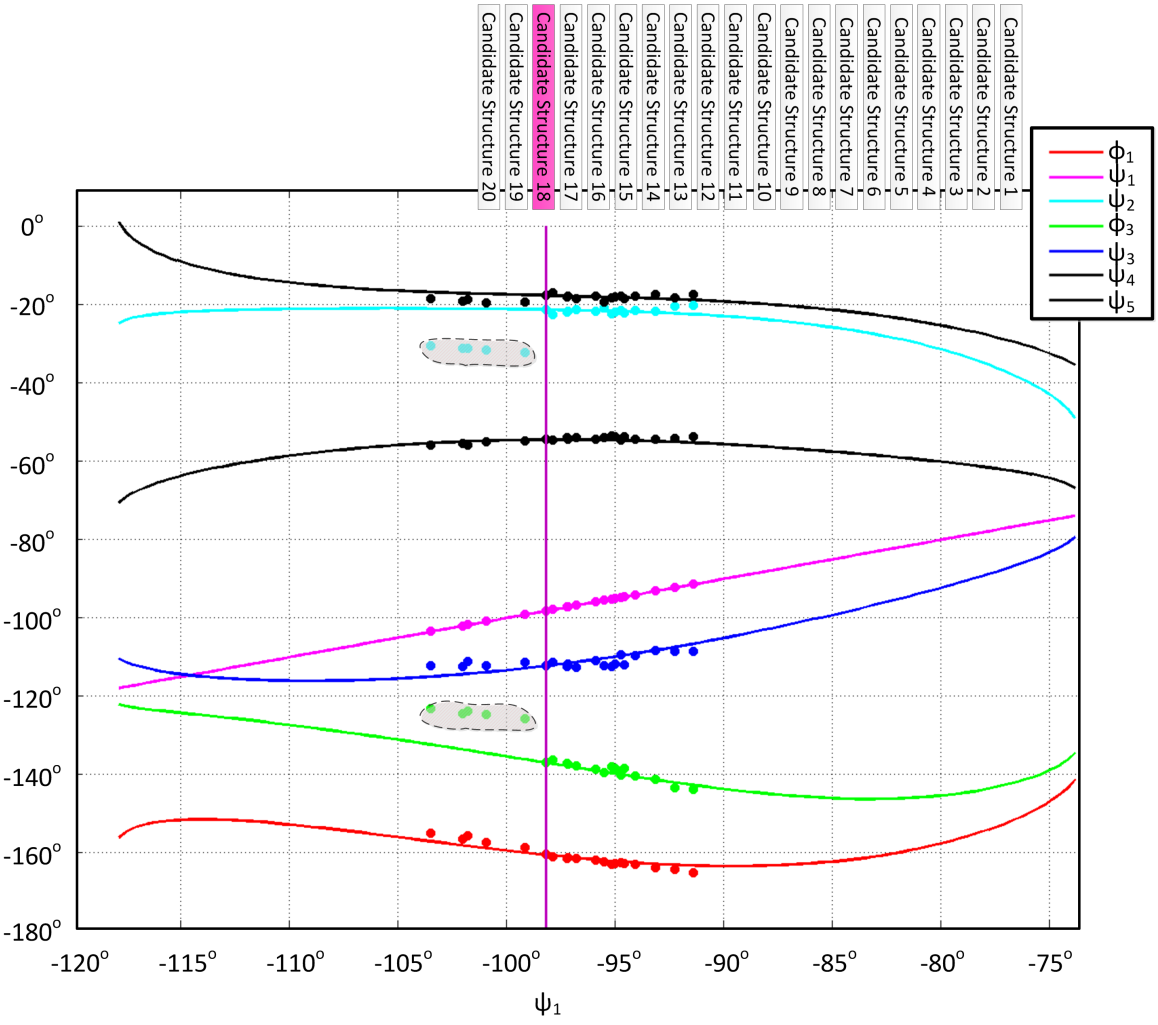


FIGURE 3.3.5: Variations in variable dihedral angles as functions of change in control variable (speculated vs actual)

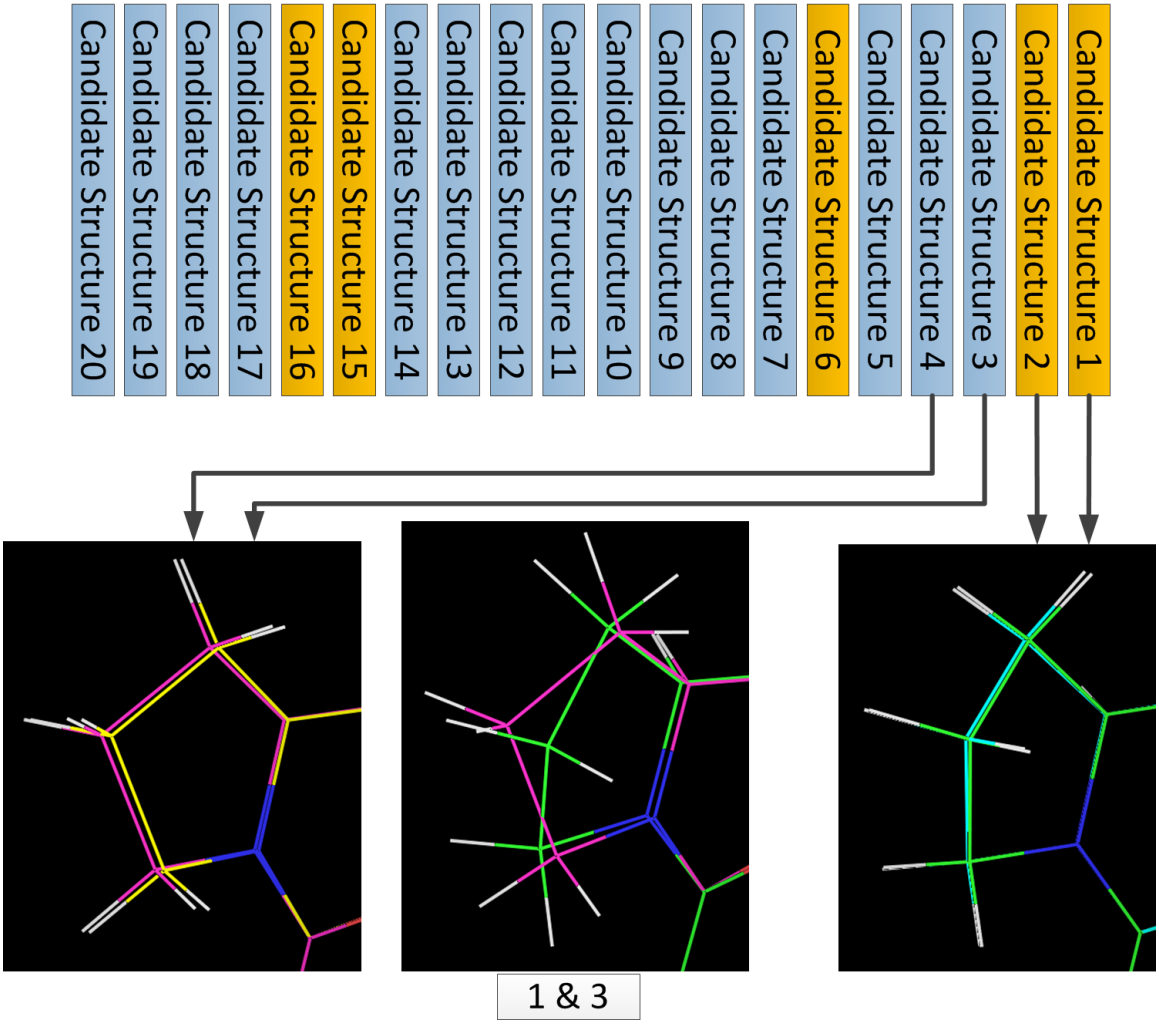


FIGURE 3.3.6: Right image shows the configuration for the group associated with off-the-curve dots. Left image shows the configuration for the rest of the candidate structures. In the middle image, the two configurations are overlapped to illustrate the difference. Images are produced with [137]

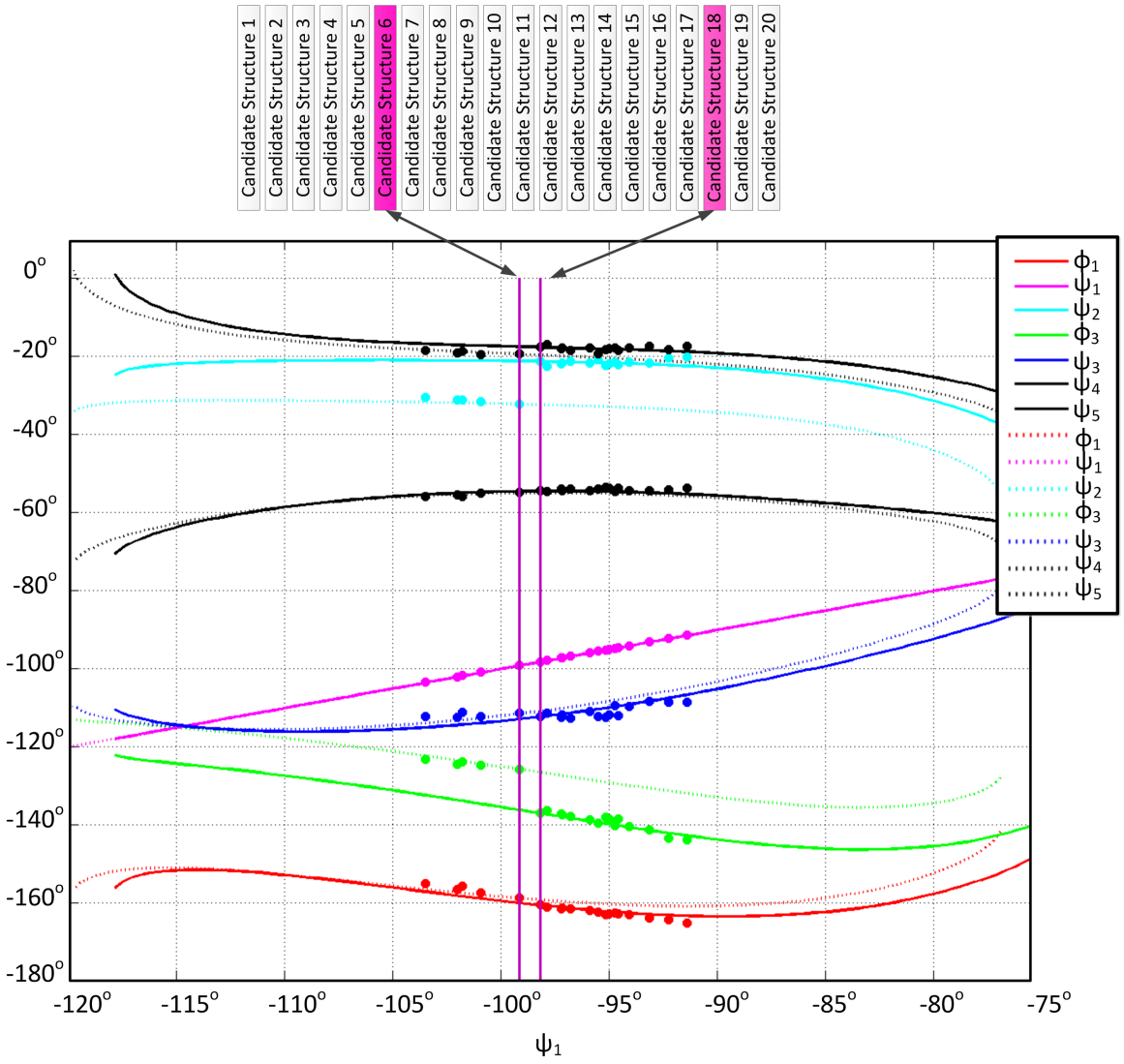


FIGURE 3.3.7: Two sets of curves describing the motions.

Candidate	GAFF	Ghemical	MMFF94	UFF
1	8093.69	537.309	589.822	1151.69
2	8096.86	550.862	591.454	1164.82
3	8107.89	526.981	574.81	1122.71
4	8105.65	544.073	575.795	1145.54
5	8111.35	540.671	577.121	1135.43
6	8087.48	536.638	588.545	1153.59
7	8111.43	539.482	578.155	1134.27
8	8114.04	546.158	578.214	1145.7
9	8111.04	543.857	574.069	1148.85
10	8104.16	531.685	577.048	1125.29
11	8114.25	543.728	576.724	1142.64
12	8109.47	532.821	575.465	1127.19
13	8110.09	533.206	576.538	1126.2
14	8109.86	536.02	577.588	1129.85
15	8097.51	553.449	592.501	1167.42
16	8093.97	546.44	592.238	1163.54
17	8108.36	530.823	574.974	1126.32
18	8117.03	544.796	579.021	1137.81
19	8115.7	546.463	576.662	1146.48
20	8115.05	543.398	579.157	1137.23

FIGURE 3.3.8: The energy values of the 20 candidate structures in KJ/mol using 4 different force fields.

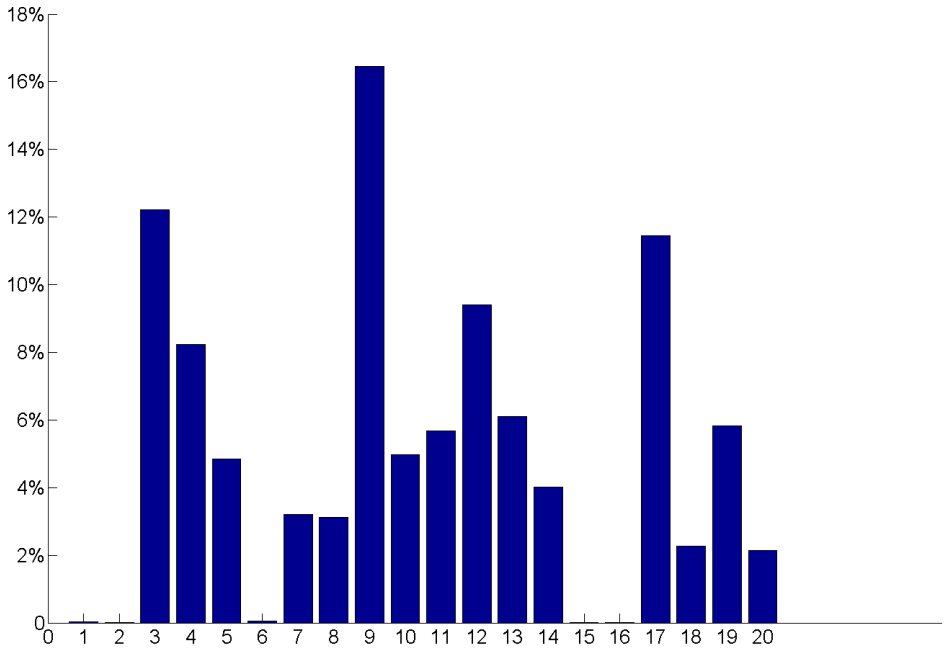


FIGURE 3.3.9: Population distribution of structure candidates.

reflect temperature and Boltzmann constant respectively), the population distribution for the 20 different conformations is obtained. The parameters were chosen as follows: $T = 300K$ and $k = 0.0083144621KJ/(molK)$. For the force field $MMFF94$, the population distribution for the 20 candidate structures can be seen in Fig. 3.3.9. The population distribution supports existence of multiple conformations in the solution. Figures 3.3.10 and 3.3.11 show the potential energy profiles (computed by our package using AMBER force field [135]) vs. control variable ψ_1 throughout the predicted motion using the constructional links of candidate structures 6 and 18 respectively. The vertical solid and dotted lines reflect the 20 candidate conformations from NMR experiments. The placement of almost all of the vertical lines within the plateau-like regions of the energy profiles imply that all of them are likely to exist in the solution.

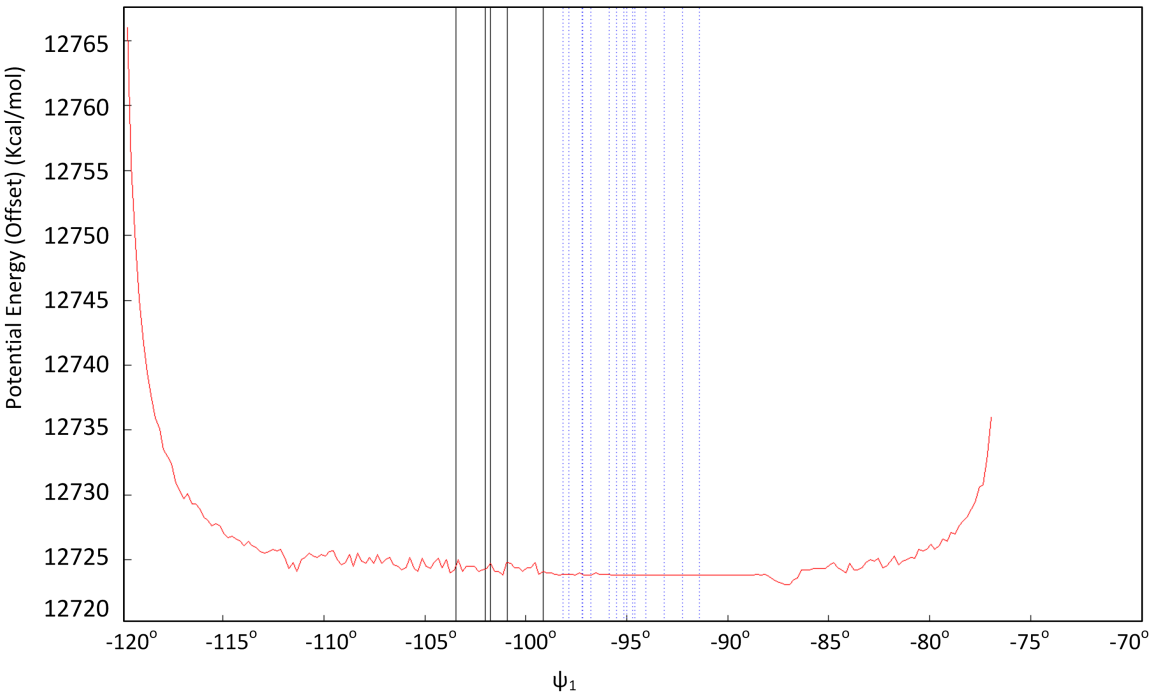


FIGURE 3.3.10: Potential energy profile vs control variable ψ_1 throughout the predicted motion using the constructional links of candidate structure 6. The vertical solid and dotted lines reflect the conformations of the candidate structures of group $\{1, 2, 6, 15, 16\}$ and group $\{3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 17, 18, 19, 20\}$ respectively.

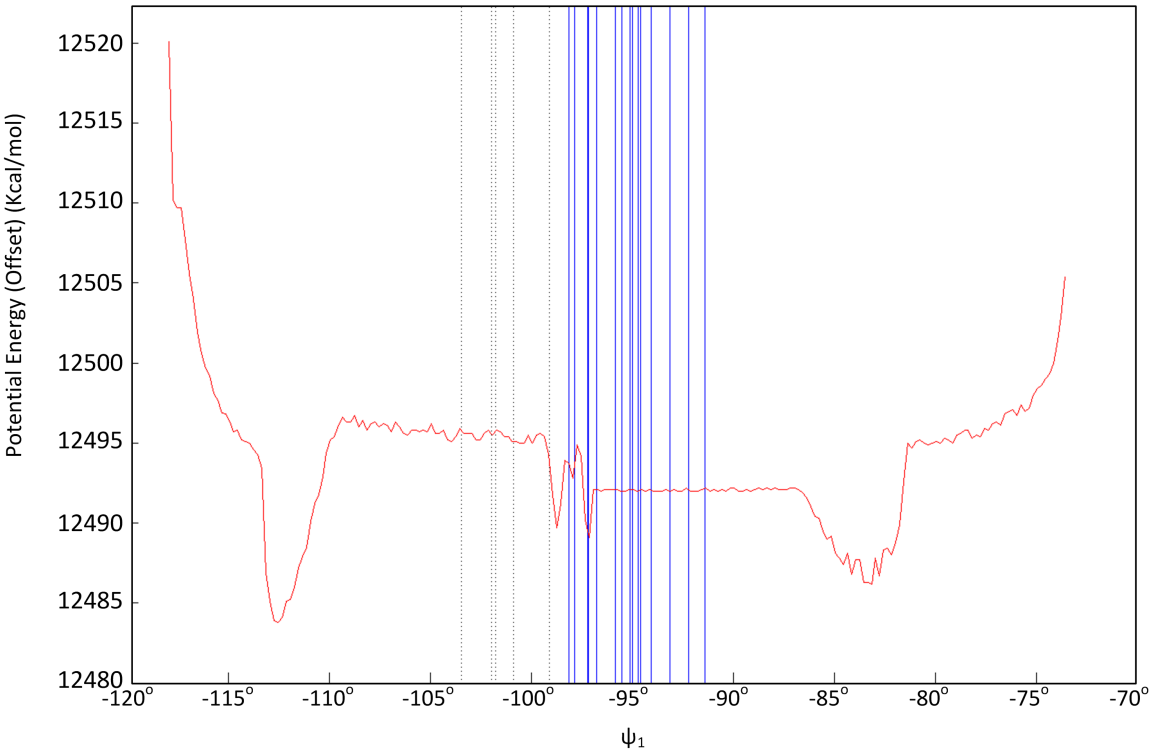


FIGURE 3.3.11: Potential energy profile vs control variable ψ_1 throughout the predicted motion using the constructional links of candidate structure 18. The vertical solid and dotted lines reflect the conformations of the candidate structures of group $\{3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 17, 18, 19, 20\}$ and group $\{1, 2, 6, 15, 16\}$ respectively.

Chapter 4

PROTOFOLD: Protein Folding Prediction Package

4.1 Introduction

Proteins are large biomolecules that are responsible for a vast array of biological functions inside the cell, and appear in the form of enzymes, antibodies, motor proteins, transport proteins, etc. [91]. The function of a protein strongly depends on its 3D structure (i.e., 'conformation') which in turn can be directly determined from the linear sequence of amino acids (AAs) linked together to form the protein chain (i.e., 'configuration') [7].[1] Therefore, the computer-aided prediction of the folded structure of a protein from the knowledge of its sequence (referred to as 'protein folding') is the key to understanding many biological processes in the cell. This knowledge is crucial

[1] In the robotics literature, the term configuration is typically used to describe the complete set of kinematic variables. However, the term conformation is typically used for that purpose in molecular biology.

toward the ultimate goal of modeling proper function or malfunction at molecular and cellular level (e.g., deadly diseases such as cancer, Alzheimer's, Parkinson's, etc.) and is central to a variety of bioengineering applications including 'protein engineering' [32, 167].

4.1.1 Related Work

There are several different computational approaches for protein folding prediction, ranging from knowledge-based techniques to methods starting from physical principles [48].

Knowledge-Based Methods. The knowledge-based approaches predict the structure of a given protein using the information extracted from previously determined structures and known types of folds. They are generally more reliable than physics-based methods, but have limited applicability in predicting new types of folds. Examples of knowledge-based techniques are homology or comparative modeling [34, 90, 106] and fold recognition or threading [19, 63, 112]. We refer the reader to [48] for a comprehensive review of such methods.

Physics-Based Methods. On the other hand, methods that utilize models formulated empirically or obtained from physical principles are less reliable and more time-consuming, but apply to a wider range of folding simulations [48]. These methods range from *de novo* [20, 138] to *ab initio* [18, 122] prediction techniques. Here we briefly review some of the common *ab initio* approaches, namely sampling methods and MD simulations [48].

Sampling methods generate numerous samples in the conformation space, followed by an evaluation of their free energies. Different search algorithms are used to find the unchallenged global minimum of the free energy, assumed to be associated with the native structure according to the 'thermodynamic hypothesis' [7]. These search methods include simulated annealing [71, 148], basin hopping [41, 96, 116, 127], evolutionary algorithms [139, 140, 170] and MC simulation with biased moves [1, 2, 28]. A review of conformation sampling methods for protein folding can be found in [88]. Sampling methods have two major limitations, namely 1) they do not provide any information about the biological pathway; and 2) finding the global minimum is not guaranteed because of the finite number of samples.

On the other hand, MD approaches simulate the biological pathway using a model built upon physical principles. Standard MD techniques include Newtonian dynamics [14, 62, 74, 156], Langevin dynamics [35, 68, 131, 169] and Brownian dynamics [6, 56, 59, 133]. A review of MD simulation methods for protein folding is provided in [136]. In order to keep the numerical algorithms stable, very small time steps (in the order of femtoseconds) along the simulation trajectory are required, which does not support folding simulation of typical proteins that span milliseconds except for small molecules [136].

Kinetostatic Compliance Method. The KCM was introduced in [82, 86] to overcome some of the key challenges in the aforementioned approaches. In this method, implemented in the software package Protofold [83–85], the protein chain is modeled as a kinematic linkage which complies under the kinetostatic effect of the force-field obtained form intramolecular interactions between the atoms. The key

contributions of KCM were

1. modeling the constrained kinematics of the protein chain with significantly fewer degrees of freedom (DOF) than, for example, those of the 'beads and springs' model used in many MD methods; and

2. converging faster to the minimum energy state by using kinetostatic (i.e., 1st-order) variations rather than dynamic (i.e., 2nd-order) response.

In KCM, each rotatable joint, used to model the constrained motion of the chemical bonds, changes by an amount proportional to the effective torque on that joint. It was shown that KCM is a faster and more stable alternative to the traditional dynamic simulation techniques [85]. The Protofold platform has since provided a kinematic testbed for subsequent research activities. Examples are predicting hydrogen bond connectivity sub-graphs [146], its application to the design of stable peptide nano-particles [145], the analysis of protein mobility (using the 'pebble game' algorithm) [144], the development of mechanical models for secondary structural elements [143], and nano-machanism synthesis from a 'link soup' of pre-specified structural elements [157, 162].

In the earlier stages of the development, the energetics were limited to intramolecular interactions in the gas-phase of the protein—e.g., Coulombic and van der Waals forces exchanged between atoms of the protein itself, ignoring the interactions with solvent molecules. However, an important class of biologically significant proteins are water-soluble, whose folding process is predominantly driven by the interactions with the solvent, particularly the so-called 'hydrophobic effect'[2] which was missing from Protofold I [85].

[2]The hydrophobic effect is explained as the strong tendency of nonpolar sidechains to pack together to form a hydrophobic core protected from the solvent by a hydrophilic surface [91]. This

Computing Solvation Effects. From a computational perspective, the solvation effects can be modeled in a number of different ways, broadly classified into 'explicit' and 'implicit' techniques.

The explicit methods use all-atom force field models such as SPC, SPC/E, TIP3P, TIP5P [80, 107], or coarse-grained (CG) models [76, 172] which are less structured representations of the solvent obtained by mapping two or more particles onto a single interaction site [107]. A prohibitive computational cost is associated with the large number of solvent molecules required to model a bulk solution.

Alternatively, approximate schemes that include the solvent effects implicitly can provide useful quantitative estimates, yet remain computationally inexpensive [134]. The implicit models estimate the contribution of each solvent-exposed atom to the solvation free energy using empirical formulae, most commonly expressed as a linear function of the solvent accessible surface area (SASA) [50]. An exact computation of SASA requires obtaining the surface area of the envelope of overlapping spheres [132], which is computationally expensive. Alternatively, approximate formulations have been developed to efficiently predict the *expected* (i.e., probabilistic average) SASA based on simplifying assumptions on the distribution of the coordinates of atoms (or groups of atoms) in the 3D space. For instance, CHARMM [21] uses the probabilistic approach from [180], which estimates the SASA as a function of the distances between pairs of atoms or residues. A similar model with similar parametrization [55] was used in GROMOS [168], a recent improvement of which was given in [3]. AMBER [38,174] uses a fast linear combination of pairwise overlaps (LCPO) algorithm [175], which improves the method in [180] by adding more terms to the approximation. Although

effect is formulated in terms of entropic changes in the solvent molecules surrounding the protein surface.

being widely popular in well-known MD packages, these methods rely on simplifying assumptions that compromise accuracy. For example, the method in [180] assumes a uniform random spatial distribution of atoms or residues, which introduces bias into the simulation results.

The inclusion of the solvation free energy computed using an adequately accurate evaluation of the SASA results in a more realistic energy- and force-model for simulating the natural behavior of water-soluble proteins.

4.1.2 Outline & Contributions

In Section 4.2, we introduce an improved free energy model making use of the linear implicit model given in [50] to compute the solvation free energy- and force-field from a knowledge of the SASA and its gradient for individual atoms at a given 3D conformation. We develop a simple offset surface enumeration technique that can approximate the SASA and its gradient up to any desired accuracy. Our method is significantly more accurate than the probabilistic methods such as those given in [175, 180] yet notably faster than the exact method given in [132], while the trade-off between speed and accuracy can be adjusted by a proper choice of the enumeration (i.e., surface sampling) density.

A second major contribution of this work is to develop significantly more efficient algorithms and data structures in Section 4.3 to speed up the computations, and to implement them into Protofold II:

1. We use a 3D hash table data structure based on a uniform spatial grid that supports fast queries to identify pairs of proximal atoms. This helps speeding up the computations by eliminating negligibly small interactions associated with

pairs of atoms that are farther than a so-called 'cut-off distance'.

2. We use a tree-based data structure to span the protein chain efficiently for the purpose of characterizing interaction types between pairs of atoms based on their distances along the topological structure of the chain.

3. We develop sequential and parallel surface enumeration algorithms to compute the SASA and its gradient for individual atoms needed for the solvation energy and force computations, respectively, up to desired accuracy.

4. We employ prefix sums [37] to compute the joint torques on the kinematic linkage of the protein chain, given the resultant forces on the individual atoms.

As a result, the numerical complexity for each KCM cycle, including the computation of resultant forces on the atoms and the links (except those resulting from solvation effects) and their conversion to joint torques, is reduced from $O(n^2)$ in Protofold I [85] to $O(n)$ in Protofold II, where n is the number of atoms in the protein molecule.[3]

The SASA evaluations for solvation force computations in our model turns out to be the bottleneck to the entire simulation—up to several orders of magnitude slower than the electrostatic and van der Waals force computations. Fortunately, the surface enumeration algorithm lends itself well to high-throughput data parallelism. In Section 4.4 we first present the CPU-parallel implementation using OpenMP, leading to moderate speed-up factors (up to one order of magnitude). To leverage the massive processing power offered by the single-instruction multiple-thread (SIMT) architecture of the modern graphics hardware—onto which our data-parallel SASA enumeration algorithm maps perfectly—we present a GPU-parallel implementation

[3]This is only the case under certain assumptions given in the subsequent sections, which are relatively reasonable for practical cases.

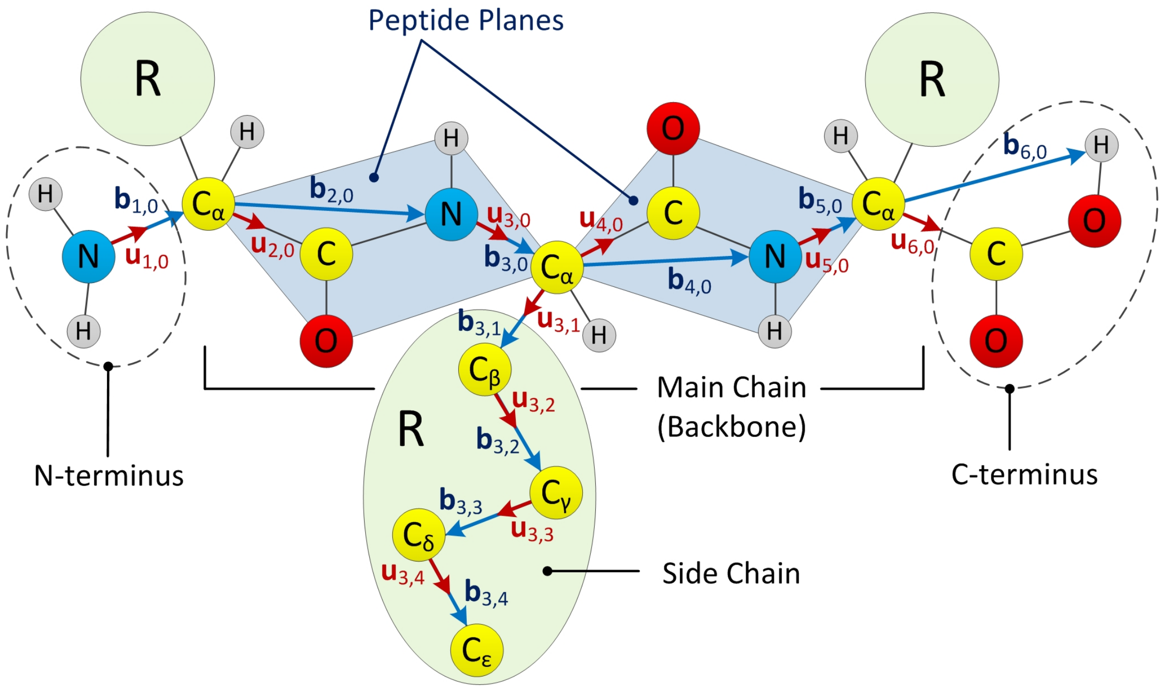


FIGURE 4.1.1: The polypeptide chain is modeled as a kinematic linkage, in which the peptide planes are assumed to be rigid.

and its optimization. The implementation takes advantage of the device memory hierarchy and hiding memory access latencies, in turn leading to larger speed-ups (up to two orders of magnitude).

4.2 Formulation

Section 4.2.1 starts with an overview of the underlying kinematic principles of the KCM simulation first introduced in [82–86]. The protein chain is modeled as an open kinematic linkage with reduced DOF in terms of dihedral and rotamer angles, which complies under the effect of interatomic and solvation forces. Next, the energy- and force-field formulation used in Protofold II is described in Section 4.2.2, with special

emphasis on the newly introduced solvation effects. Lastly, the KCM optimization process is presented in Section 4.2.3.

4.2.1 Kinematic Model

Proteins are long polymeric chains made of AAs, which exist in only 20 different types (except for few rare exceptions), joined together as a linear polypeptide chain [91], structural details of which are summarized in Appendix A. Here we restrict ourselves to the kinematic representation of the chain's conformation within the scope of KCM.

Linkage Parameterization. Figure 4.1.1 schematically illustrates the repetitive sequence of $-\mathrm{N}-\mathrm{C}_\alpha-\mathrm{C}-$ atoms,[4] called the 'backbone' or the 'main chain', with 'side chains' resembling branches that extend out of it. As explained in Appendix A, the backbone conformation can be specified to an adequate accuracy by two sets of dihedral angles; namely,

- $-180^\circ \leq \phi_i < +180^\circ$ (around $\mathrm{N}-\mathrm{C}_\alpha$ in AA_i); and
- $-180^\circ \leq \psi_i < +180^\circ$ (around $\mathrm{C}_\alpha-\mathrm{C}$ in AA_i);

for $1 \leq i \leq m$, where m is the number of AA residues along the chain. The conformation of each side chain, on the other hand, can be specified by up to 4 extra angles $-180^\circ \leq \chi_{i,k} < +180^\circ$ for $1 \leq k \leq l_i$ where $0 \leq l_i \leq 4$ is the number of side chain links of AA_i, and the subscript k corresponds to the bonds numbered in the obvious order along the side chain C and N atoms.

[4] Hereon, the notations C_α and C correspond to the alpha-carbon and carboxyl-carbon, respectively.

To set a reference for the angle measurements, the zero-reference position description (ZRPD) method [69] is used. The zero-position (ZP) for the protein chain is defined as the conformation of the serial linkage in which all peptide planes are coplanar (i.e., $\phi_i^0 = \psi_i^0 = -180°$) and side chain dihedrals are set to default low energy values identified as 'rotamers' [86].

To unify the notations, all angular variables are denoted by $\theta_{j,k}$ $(1 \leq j \leq 2m, 0 \leq k \leq 4)$ where

$$\theta_{2i-1,0} = \phi_i + 180°, \quad 1 \leq i \leq m, \tag{4.2.1}$$

$$\theta_{2i,0} \quad = \psi_i + 180°, \quad 1 \leq i \leq m, \tag{4.2.2}$$

$$\theta_{2i-1,k} = \chi_{i,k} - \chi_{i,k}^0, \quad 1 \leq i \leq m,\ 1 \leq k \leq l_i \leq 4, \tag{4.2.3}$$

where $0° \leq \theta_{j,k} < 360°$. The shifts in (4.2.1) and (4.2.2) by the intercept values of $\phi_i^0 = \psi_i^0 = -180°$ and in (4.2.3) by the favorable rotamer angles $\chi_{i,k}^0$ ensure $\theta_{j,k} = 0°$ at the ZP conformation.

A similar indexing scheme is used to identify the unit vectors along the rotation axes of revolute joints associated with these angles denoted by $\mathbf{u}_{j,k}$ $(1 \leq j \leq 2m, 0 \leq k \leq 4)$, i.e.,

- $\mathbf{u}_{2i-1,0}$ $(1 \leq i \leq m)$ is the unit vector along the bond between N of AA_i and C_α of AA_i;
- $\mathbf{u}_{2i,0}$ $(1 \leq i \leq m)$ is the unit vector along the bond between C_α of AA_i to the C of AA_i; and
- $\mathbf{u}_{2i-1,k}$ $(1 \leq i \leq m, 1 \leq k \leq 4)$ are the unit vectors along the successive side chain C and N atoms.

Thus the kinematics of the linkage—which abstracts the protein conformation—can be completely specified in terms of the rigid body rotation transformations obtained from these rotation angles and rotation axes.

The spatial orientation of the rigid peptide planes can be described conveniently with a pair of base vectors whose linear combination spans the peptide plane. The so-called 'body vectors' are denoted by $\mathbf{b}_{j,k}$ $(1 \leq j \leq 2m, 0 \leq k \leq 4)$, i.e.,

- $\mathbf{b}_{2i-1,0}$ $(1 \leq i \leq m)$ is the base vector that connects the N of AA_i to the C_α of AA_i;
- $\mathbf{b}_{2i,0}$ $(1 \leq i \leq m)$ is the base vector that connects the C_α of AA_i to the N of AA_{i+1}; and
- $\mathbf{b}_{2i-1,k}$ $(1 \leq i \leq m, 1 \leq k \leq 4)$ are the base vectors along the successive side chain C and N atoms.

The first two sets of vectors listed above are called the 'main chain body vectors'. Every vector in the peptide plane that describes the relative positions of any two atoms can be obtained as a linear combination of these base vectors as $C_1\mathbf{b}_{2i,0} + C_2\mathbf{b}_{2i+1,0}$. The coefficients C_1 and C_2, referred to as 'peptide plane constants', are invariant with respect to the rotations in the chain, thus can be precomputed prior to the KCM simulation. Different pairs of coefficients are used for vectors describing the relative positions of different pairs of atoms. Based on experimental evidence, it is a reasonable assumption that these coefficients are the same across all AAs [155], and the average values are given in Table 4.2.1.[5]

[5]Nevertheless, in Protofold II the user has the option to choose whether to use the values provided in Table 4.2.1 for all AAs, or to maintain the refined values when available—e.g., when the protein is imported from the protein data-bank (PDB).

TABLE 4.2.1: Peptide plane constants for bond vectors [155].

BV	C_1	C_2	BV	C_1	C_2
$\overrightarrow{C_\alpha C}$	-0.2761	$+1.4488$	$\overrightarrow{CO}$	-1.3324	$+2.3401$
$\overrightarrow{CN}$	$+1.2761$	-1.4488	$\overrightarrow{NH}$	$+1.4103$	-2.5111

In addition to the main chain body vectors, the 'side chain body vectors' (the third group listed above) are defined for the relative positions of the C and N atoms along the side chains. We refer the reader to [155] for more details about the molecular model of the peptide unit.

For a protein chain with m AA residues, the number of links can be obtained as

$$l = \left(2m + \sum_{i=1}^{m} l_i \right) \leq 6m = O(m), \tag{4.2.4}$$

noting that $l_i \leq 4$. The term $2m$ counts two rigid links per each AA's backbone—one for $-$CO$-$NH$-$ and one for $-C_\alpha-$ in the peptide unit—in order to have each rigid link connected to the next with a single revolute joint along either N$-C_\alpha$ or $C_\alpha-$C, as depicted in Fig. 4.1.1. The second term accounts for the number of additional side chain links. As a result, the total DOF of the kinematic linkage is equal to the number of links. Table 4.2.2 gives a complete description of dihedral angles, unit vectors, and body vectors for the entire protein chain.

Kinematic Equations. The instantaneous conformation of the protein chain is related to the ZP conformation with a set of rigid body transformations. Given a particular conformation in terms of $\theta_{j,k}$ $(1 \leq j \leq 2m, 0 \leq k \leq 4)$, the unit vectors

TABLE 4.2.2: Kinematic variables of the polypeptide linkage.

Symbol	Description
$\theta_{2i-1,0}$	Torsion angle ϕ_i around main chain N$-$C$_\alpha$ in AA_i
$\mathbf{u}_{2i-1,0}$	Unit vector along main chain N$-$C$_\alpha$ in AA_i
$\mathbf{b}_{2i-1,0}$	Body vector from N to C$_\alpha$ in AA_i
$\theta_{2i,0}$	Torsion angle ψ_i around main chain C$_\alpha-$C in AA_i
$\mathbf{u}_{2i,0}$	Unit vector along main chain C$_\alpha-$C in AA_i
$\mathbf{b}_{2i,0}$	Body vector from C$_\alpha$ in AA_i to N in AA_{i+1}[†]
$\theta_{2i-1,k}$	Torsion angle $\chi_{i,k}$ of side chain C/Ns in AA_i
$\mathbf{u}_{2i-1,k}$	Unit vector along side chain C/Ns in AA_i
$\mathbf{b}_{2i-1,k}$	Body vector connecting side chain Cs in AA_i

[†] The exception is $\mathbf{b}_{2m,0}$ which connects C$_\alpha$ to the carboxyl H in AA_{2m}.

and body vectors are transformed as follows:

$$[\mathbf{u}_{j,k}] = [M_{j,k}][\mathbf{u}^0_{j,k}], \quad 1 \le j \le 2m,\ 0 \le k \le 4, \tag{4.2.5}$$

$$[\mathbf{b}_{j,k}] = [M_{j,k}][\mathbf{b}^0_{j,k}], \quad 1 \le j \le 2m,\ 0 \le k \le 4, \tag{4.2.6}$$

where the superscript 0 indicates the reference ZP conformation. $[M_{j,k}]$ is the 3×3 matrix representation of the rigid body transformation $M_{j,k} \in \mathrm{SO}(3)$ that maps the ZP unit and body vectors $\mathbf{u}^0_{j,k}$ and $\mathbf{b}^0_{j,k}$ to their transformed orientations $\mathbf{u}_{j,k}$ and $\mathbf{b}_{j,k}$, respectively. These vectors are expressed using 3×1 column matrices. The transformation matrix for the main chain vectors ($k = 0$) can be calculated as a

product of successive rotations around individual joints in the main chain:

$$[M_{j,0}] = \prod_{r=1}^{j} [R_{r,0}], \quad 1 \leq j \leq 2m, \tag{4.2.7}$$

while the transformation matrix for the side chain vectors ($k \geq 1$) is defined as a product of rotations around joints in the main chain, and those around the side chain:

$$[M_{2i-1,k}] = \prod_{r=1}^{2i-1} [R_{r,0}] \prod_{s=1}^{k} [R_{2i-1,s}], \quad 1 \leq i \leq m,\ 1 \leq k \leq 4, \tag{4.2.8}$$

where $[R_{r,s}]$ is the 3×3 matrix representation of the joint rotation transformation $R_{r,s} \in \mathrm{SO}(3)$ around the ZP unit vector $\mathbf{u}^0_{r,s}$ with an angle $\theta_{r,s}$ ($1 \leq r \leq i, 0 \leq s \leq k$) [86], using the right-hand rule to choose the direction.

Once the body vectors are obtained using (4.2.6) for a given conformation, the moved atom center positions can be computed for the individual atoms. Assuming that the N atom at the amino-terminus is fixed at the origin, the coordinates of the main chain N and C_α atoms are obtained as

$$[\mathbf{r}_{j,0}] = \sum_{r=1}^{j} [\mathbf{b}_{r,0}], \quad 1 \leq j \leq 2m-1, \tag{4.2.9}$$

where the index $j = 2i-1$ corresponds to the C_α atom of residue AA_i while the index $j = 2i$ corresponds to the N atom of the residue AA_{i+1} for $1 \leq i \leq m$. The coordinates for the other atoms in the peptide group, namely H, C and O, are obtained from those for C_α and N, and a linear combination $C_1\mathbf{b}_{2i,0} + C_2\mathbf{b}_{2i+1,0}$ of main chain body vectors using the coefficients C_1 and C_2 given in Table 4.2.1. For the side chain

C and N atoms, the coordinates are similarly obtained as

$$[\mathbf{r}_{2i-1,k}] = \sum_{r=1}^{2i-1}[\mathbf{b}_{r,0}] + \sum_{s=1}^{k}[\mathbf{b}_{2i-1,s}], \quad 1 \le i \le m,\ 1 \le k \le 4, \tag{4.2.10}$$

where $k = 1, 2, 3$, and 4 corresponds to the successive side chain C and N atoms in the residue AA_i. The coordinates for all other side chain atoms are obtained similarly from vectors along the side chain bonds subjected to the same set of motions [82].

The atom position vectors obtained from (4.2.9) and (4.2.10) at each iteration are used in the next section to compute the energies, forces, and torques that will determine the motion for the subsequent iteration.

4.2.2 Force Model

The interatomic effects can be classified into covalent and noncovalent interactions. The covalent interactions need not be considered explicitly in the force-field, since they are implicitly introduced in terms of the kinematic constraints innate to the kinematic chain model adopted in Section 4.2.1. The noncovalent forces, which are responsible for conformational changes in the protein chain, can be derived from the free energy formulation that follows.

For a protein chain made of n atoms, we assign each atom with a unique index $1 \le i \le n$, and its center coordinates $\mathbf{r}_i \in \mathbb{R}^3$ obtained from dihedral angles using kinematic equations (4.2.9) and (4.2.10).[6] Each atom is identified by a tuple $a_i = (i, \mathbf{r}_i, R_i, q_i, \epsilon_i, \gamma_i, \cdots)$ $(1 \le i \le n)$, containing its index, position, radius, charge, well

[6]Note the slight change of notations from Section 4.2.1, where the subscript $j = 2i - 1$ or $2i$ referred to the AA index $1 \le i \le m$, while in Section 4.2.2 the single subscript $1 \le i \le n$ refers to the atom index.

depth parameter, solvation parameter, and other atomic constants, to be introduced shortly. The set of all atoms in the molecule is denoted by $\mathbb{A} = \{a_1, a_2, \cdots, a_n\}$. The aggregated free energy of all atoms in $\mathbb{A}$ can be decomposed into the following terms:

$$G^{\text{tot}}(\mathbb{A}) = G^{\text{elec}}(\mathbb{A}) + G^{\text{vdw}}(\mathbb{A}) + G^{\text{cav}}(\mathbb{A}), \tag{4.2.11}$$

where $G^{\text{elec}}(\mathbb{A})$ is the electrostatic energy, including intramolecular charge interactions, hydrogen bonding effects, and the induced polarization in the solvent when the molecule is dissolved. $G^{\text{vdw}}(\mathbb{A})$ is the sum of intramolecular van der Waals energies, also called 'steric effects', resulted from induced dipoles in the molecule. The sum of the first two terms has been accounted for in Protofold I [83–85] using the AMBER force-field model [174]. $G^{\text{cav}}(\mathbb{A})$ is the nonpolar solvation free energy, the free energy change resulting from transfering a molecule from vacuum to solvent, i.e., the entropic change due to the formation of the cavity occupied by the instantaneous 3D shape of the protein [9]. Experimental results have shown that many water-soluble protein folding reactions are predominantly driven by a favorable reduction in $\Delta G^{\text{cav}}(\mathbb{A})$ [91], hence we incorporated this term into the improved energy formulation for Protofold II.

Electrostatic Interactions. The charge interactions are formulated using the modified form of Coulomb's law [174]:

$$G^{\text{elec}}(\mathbb{A}) = \sum_{a_i \in \mathbb{A}} \sum_{a_j \in \mathbb{A} - \{a_i\}} \frac{1}{4\pi\varepsilon_{i,j}} \frac{q_i q_j}{d_{i,j}}, \tag{4.2.12}$$

where $d_{i,j} = \|\mathbf{r}_i - \mathbf{r}_j\|_2$ is the interatomic center distance, q_i and q_j are the electrostatic charges, and $\mathbf{r}_i$ and $\mathbf{r}_j$ are the position vectors of the pair of atoms $a_i, a_j \in \mathbb{A}$, respectively. $\varepsilon_{i,j} = \kappa_{i,j}\varepsilon_0$ is the 'dielectric constant' and is generally larger than vacuum permittivity $\varepsilon_0 \approx 8.854 \times 10^{-12}$ Farads (i.e., $\kappa_{i,j} > 1$). Thus (4.2.12) can be used to calculate the interactions between charges in the solvent, if a continuum dielectric model is used [91]. The dielectric constant reflects the ability of the environment to attenuate electrostatic interactions, e.g., $\kappa_{i,j} \sim 80$ for aqueous solvent and $\kappa_{i,j} \sim 2$–4 for the interior of the protein [91], where the larger value for the former is due to the induced polarization of water molecules. It is worthwhile noting that because of the nonuniformity of the dielectric medium, the most accurate computation of the electrostatic energy requires solving Poisson-Boltzman (PB) differential equations [53]. However, solving PB for every cycle of the KCM simulation is computationally expensive, and approximate alternatives such as generalized Born (GB) model can be used [121, 151]. A simple distance-dependent dielectric constant is used here (following the convention in [85]) to mimic the polarization effect, with closer interactions weighted more heavily [174]. The resultant Coulombic force $\mathbf{F}_i^{\text{elec}} = -\nabla_{\mathbf{r}_i} G^{\text{elec}}$ applied on the atom a_i by other atoms is then obtained as

$$\mathbf{F}_i^{\text{elec}}(\mathbb{A}) = \sum_{a_j \in \mathbb{A} - \{a_i\}} \frac{1}{4\pi\varepsilon_{i,j}} \frac{q_i q_j}{d_{i,j}^2} \mathbf{e}_{i,j}, \tag{4.2.13}$$

where $\mathbf{e}_{i,j} = (\mathbf{r}_i - \mathbf{r}_j)/d_{i,j}$ is the unit vector along the line of centers of the pair of atoms $a_i, a_j \in \mathbb{A}$. Since $\mathbf{F}_i^{\text{elec}} \propto 1/d_{i,j}^2$, electrostatic interactions between atoms that are farther than a so-called cut-off distance $d_{\text{cut}}^{\text{elec}} := 9.0\ \mathring{A}$ are usually deemed

negligible in the literature [91].[7] Therefore (4.2.13) is approximated as follows to reduce the number of pairwise computations between all atoms:

$$\mathbf{F}_i^{\text{elec}}(\mathbb{A}_i^{\text{elec}}) \approx \sum_{a_j \in \mathbb{A}_i^{\text{elec}}} \frac{1}{4\pi\varepsilon_{i,j}} \frac{q_i q_j}{d_{i,j}^2} \mathbf{e}_{i,j}, \tag{4.2.14}$$

where $\mathbb{A}_i^{\text{elec}} = \{a_j \in \mathbb{A} - \{a_i\} \mid d_{i,j} \leq d_{\text{cut}}^{\text{elec}}\}$ is referred to as the neighborhood of atom a_i associated with the electrostatic force-field, and consists of all the atoms whose distance to a_i are bounded by the cut-off distance $d_{\text{cut}}^{\text{elec}}$.

Van der Waals Interactions. The van der Waals interactions are formulated using the empirical Lennard-Jones 6-12 potential function formula [174]:

$$G^{\text{vdw}}(\mathbb{A}) = \sum_{a_i \in \mathbb{A}} \sum_{a_j \in \mathbb{A} - \{a_i\}} \epsilon_{i,j} \left[\left(\frac{D_{i,j}}{d_{i,j}} \right)^{12} - 2 \left(\frac{D_{i,j}}{d_{i,j}} \right)^{6} \right], \tag{4.2.15}$$

where $d_{i,j} = \|\mathbf{r}_i - \mathbf{r}_j\|_2$ is the interatomic center distance, $D_{i,j} = R_i + R_j$ is the 'van der Waals distance' in which R_i, R_j are the van der Waals radii of the atoms $a_i, a_j \in \mathbb{A}$, respectively. $\epsilon_{i,j} = \sqrt{\epsilon_i \epsilon_j}$ is the 'depth of potential well' for the particular pair of atoms. It is worthwhile noting that the van der Waals effects have the same origin as the electrostatic forces, and reflect the induced dipoles due to transient fluctuations in electron clouds of the interacting atoms [91]. The resultant van der Waals force $\mathbf{F}_i^{\text{vdw}} = -\nabla_{\mathbf{r}_i} G^{\text{vdW}}$ on the atom a_i by other atoms is then obtained as

$$\mathbf{F}_i^{\text{vdw}}(\mathbb{A}) = \sum_{a_j \in \mathbb{A} - \{a_i\}} 12\epsilon_{i,j} \left(\frac{D_{i,j}^{12}}{d_{i,j}^{13}} - \frac{D_{i,j}^{6}}{d_{i,j}^{7}} \right) \mathbf{e}_{i,j}, \tag{4.2.16}$$

[7] Our experiments with larger molecules show that 9.0 $\mathring{A}$ is not always a proper cut-off distance and larger values need to be used, as demonstrated in Section 4.5.3.

where $\mathbf{e}_{i,j} = (\mathbf{r}_i - \mathbf{r}_j)/d_{i,j}$ is the unit vector along the line of centers of the pair of atoms $a_i, a_j \in \mathbb{A}$. The van der Waals forces have a much smaller effect radius and are significant only when the atoms approach each other very closely. The repulsive component becomes very large when the two atoms penetrate into each other, an effect widely known as the 'steric clash'. The attractive component known as the 'London dispersion' force, on the other hand, is dominant when the atoms are farther than the van der Waals distance $D_{i,j}$ [91]. These interactions decay much faster than Coulombic forces, hence a smaller cut-off distance of $d_{\text{cut}}^{\text{vdw}} := 5.0\ \mathring{A}$ is sufficient [91] resulting in the following approximation:

$$\mathbf{F}_i^{\text{vdw}}(\mathbb{A}_i^{\text{vdw}}) \approx \sum_{a_j \in \mathbb{A}_i^{\text{vdw}}} 12\epsilon_{i,j} \left(\frac{D_{i,j}^{12}}{d_{i,j}^{13}} - \frac{D_{i,j}^{6}}{d_{i,j}^{7}} \right) \mathbf{e}_{i,j}, \tag{4.2.17}$$

where $\mathbb{A}_i^{\text{vdw}} = \{a_j \in \mathbb{A} - \{a_i\} \mid d_{i,j} \leq d_{\text{cut}}^{\text{vdw}}\}$ is referred to as the neighborhood of the atom a_i associated with the van der Waals force-field, and consists of all the atoms whose distance to a_i are bounded by the cut-off distance $d_{\text{cut}}^{\text{vdw}}$.

Interaction Classification. The interactions discussed so far are between the pairs of atoms that are *not* covalently bonded, thus (4.2.14) and (4.2.17) have to be modified to eliminate the terms corresponding to the pairs of bonded atoms (i.e., '1-2 interactions'). Furthermore, it is a common convention in molecular mechanics to modify the electrostatic and van der Waals interactions between the pairs of atoms bonded to a common atom, i.e., atoms that are 2 bonds apart along the chain (i.e., '1-3 interactions'), as well as the atoms that are 3 bonds apart along the chain (i.e., '1-4 interactions') [128]. Consequently, the empirical forms of (4.2.14) and (4.2.17)

TABLE 4.2.3: Atomic solvation parameters adopted from [177] for different adjustments by [93, 147]. Units are in $kcal\ mol^{-1}\mathring{A}^{-2}$.

Atom type	Adjustment in [93]	Adjustment in [147]
C	$+0.004 \pm 0.003$	$+0.012 \pm 0.003$
O/N	-0.113 ± 0.014	-0.116 ± 0.013
S	-0.017 ± 0.022	-0.018 ± 0.021
O^-	-0.166 ± 0.038	-0.175 ± 0.036
N^+	-0.169 ± 0.031	-0.186 ± 0.022

are modified as

$$\mathbf{F}_i^{\text{elec}}(\mathbb{A}_i^{\text{elec}}) \approx \sum_{a_j \in \mathbb{A}_i^{\text{elec}}} \frac{w_{i,j}^{\text{elec}}}{4\pi\varepsilon_{i,j}} \frac{q_i q_j}{d_{i,j}^2} \mathbf{e}_{i,j}, \tag{4.2.18}$$

$$\mathbf{F}_i^{\text{vdw}}(\mathbb{A}_i^{\text{vdw}}) \approx \sum_{a_j \in \mathbb{A}_i^{\text{vdW}}} 12 w_{i,j}^{\text{vdw}} \epsilon_{i,j} \left(\frac{D_{i,j}^{12}}{d_{i,j}^{13}} - \frac{D_{i,j}^{6}}{d_{i,j}^{7}} \right) \mathbf{e}_{i,j}, \tag{4.2.19}$$

where $w_{i,j}^{\text{elec}}$ and $w_{i,j}^{\text{vdw}}$ are the weight factors for the electrostatic and van der Waals forces, respectively, for the pair of atoms $a_i, a_j \in \mathbb{A}$ depending on their interaction type. The weights are set to $w_{i,j} = 0$ for 1-2 interactions, and $0 \leq w_{i,j} \leq 1$ for 1-3 and 1-4 interactions, whose values vary across different force models [21, 38, 168, 174]. $w_{i,j} = 1$ for all other situations. In other words, the atoms that have at least 4 bonds in between them along the graph of covalent bonds are far enough to be considered unaffected by the covalent electron clouds, as originally formulated in (4.2.14) and (4.2.17).

Nonpolar Solvation Effects. The hydrophobic free energy of solvation, which reflects the entropy changes in the solvent molecules due to cavity creation, is formu-

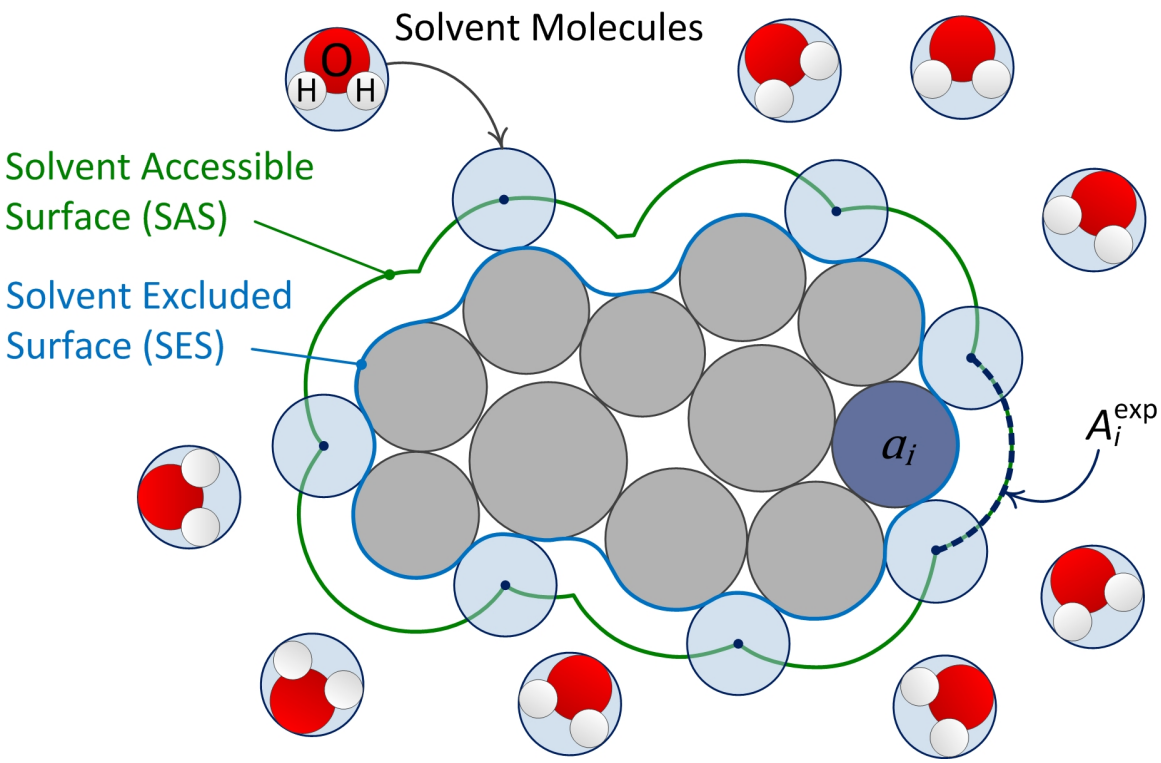


FIGURE 4.2.1: The solvent-accessible and -excluded surfaces.

lated using the linear empirical formulation in [177]:

$$G^{\text{cav}}(\mathbb{A}) = \sum_{a_j \in \mathbb{A}} \gamma_j \; A_j^{\text{exp}}, \tag{4.2.20}$$

where γ_j is the atomic solvation parameter and A_j^{exp} is the Lee and Richards SASA for the atom $a_j \in \mathbb{A}$ [94]. To obtain the atomic SASA at a given snapshot, a probe radius of $R_{\text{H}_2\text{O}} = 1.2 - 1.4 \; \mathring{A}$ is used to offset the van der Waals surfaces of the individual atoms as illustrated in Fig. 4.2.1. These offset spheres are overlapped to obtain the contributions of different atoms to the total SASA. The atomic solvation parameter γ_j is an estimate of the free energy per unit area required to transfer the atom from vacuum to water, and depends on the atom type [177]. Table 4.2.3 shows the values of this parameter for different atom types (namely, C, uncharged O or N, S, O^-, and N^+) obtained in [177] based on the experimental results in [182] adjusted by [93, 147]. The hydrophobic interaction forces $\mathbf{F}_i^{\text{cav}} = -\nabla_{\mathbf{r}_i} G^{\text{cav}}$ on the atom a_i by other atoms is then obtained as

$$\mathbf{F}_i^{\text{cav}}(\mathbb{A}) = -\sum_{a_j \in \mathbb{A}} \gamma_j \nabla_{\mathbf{r}_i} A_j^{\text{exp}}, \tag{4.2.21}$$

where $\nabla_{\mathbf{r}_i} A_j^{\text{exp}}$ is the gradient of the exposed area of the atom a_j due to an infinitesimal displacement of a_i. It is important to note that, unlike the force formulae presented earlier for the electrostatic and van der Waals effects in (4.2.13) and (4.2.16), the summation in (4.2.21) for the solvation free energy gradient iterates over all $a_j \in \mathbb{A}$ *including* a_i itself.

Considering the case when $i = j$, one realizes that displacing the atom a_i in any direction has the same effect on the geometry of the protein surface as displacing all

the other atoms in the opposite direction. Therefore

$$\nabla_{\mathbf{r}_i} A_i^{\exp} = - \sum_{a_j \in \mathbb{A} - \{a_i\}} \nabla_{\mathbf{r}_j} A_i^{\exp}. \tag{4.2.22}$$

Substituting for this term in (4.2.21) leads to the following symmetric form, whose range of summation excludes a_i itself, similar to (4.2.13) and (4.2.16):

$$\begin{aligned} \mathbf{F}_i^{\text{cav}}(\mathbb{A}) &= -\gamma_i \nabla_{\mathbf{r}_i} A_i^{\exp} - \sum_{a_j \in \mathbb{A} - \{a_i\}} \gamma_j \nabla_{\mathbf{r}_i} A_j^{\exp} \\ &= \sum_{a_j \in \mathbb{A} - \{a_i\}} \left(\gamma_i \nabla_{\mathbf{r}_j} A_i^{\exp} - \gamma_j \nabla_{\mathbf{r}_i} A_j^{\exp} \right). \end{aligned} \tag{4.2.23}$$

We show in Section 4.4 that (4.2.23) is computationally preferable over (4.2.21). To further simplify (4.2.23), note that for a pair of atoms a_i and a_j the infinitesimal displacement of one does not affect the overlapped solvent exposed area of the other if their offset spheres (i.e., the spheres with radii $R_i^{\text{off}} = R_i + R_{\text{H}_2\text{O}}$ and $R_j^{\text{off}} = R_j + R_{\text{H}_2\text{O}}$, respectively) do not intersect, i.e., $\nabla_{\mathbf{r}_i} A_j^{\exp} = \nabla_{\mathbf{r}_j} A_i^{\exp} = 0$ if $d_{i,j} > R_i + R_j + 2R_{\text{H}_2\text{O}}$. Therefore, the number of terms that contribute a nonzero value to the summation of (4.2.23) is significantly reduced:

$$\mathbf{F}_i^{\text{cav}}(\mathbb{A}_i^{\text{cav}}) = \sum_{a_j \in \mathbb{A}_i^{\text{cav}}} \left(\gamma_i \nabla_{\mathbf{r}_i} A_j^{\exp} - \gamma_j \nabla_{\mathbf{r}_j} A_i^{\exp} \right), \tag{4.2.24}$$

where $\mathbb{A}_i^{\text{cav}} = \{a_j \in \mathbb{A} - \{a_i\} \mid d_{i,j} \leq R_i + R_j + 2R_{\text{H}_2\text{O}}\}$ is referred to as the neighborhood of the atom a_i associated with the nonpolar solvent effects. For practical reasons that will be explained in Section 4.3.2, we use a larger neighborhood redefined as $\mathbb{A}_i^{\text{cav}} = \{a_j \in \mathbb{A} - \{a_i\} \mid d_{i,j} \leq d_{\text{cut}}^{\text{cav}}\}$ using the more conservative (but

constant across all pairs of atoms) cut-off distance of $d_{\text{cut}}^{\text{cav}} := 2(R_{\max} + R_{\text{H}_2\text{O}})$, where $R_{\max} = \max_{a_i \in \mathbb{A}}\{R_i\}$. A value of $d_{\text{cut}}^{\text{cav}} := 8\mathring{A}$ is typically safe. Note that unlike the case with (4.2.13) and (4.2.16), eliminating pairwise interactions with $d_{i,j} > 8\mathring{A}$ from (4.2.24) does not impart an approximation error.

4.2.3 Kinetostatic Simulation

We use the KCM (presented in [82–86] for protein folding) to explicitly integrate the conformational changes of the linkage model under the kinetostatic effect of the force-field computed in Section 4.2.2.

Link Forces and Torques. For a protein chain with a total of $l = O(m)$ links, where m is the number of AA residues, the resultant force and torque applied to the j^{th} link ($1 \leq j \leq l$) are computed as

$$\mathbf{F}_j^{\text{link}} = \sum_{a_i \in \mathbb{L}_j} \left(\mathbf{F}_i^{\text{elec}} + \mathbf{F}_i^{\text{vdw}} + \mathbf{F}_i^{\text{cav}}\right), \tag{4.2.25}$$

$$\mathbf{T}_j^{\text{link}} = \sum_{a_i \in \mathbb{L}_j} \mathbf{r}_i \times \left(\mathbf{F}_i^{\text{elec}} + \mathbf{F}_i^{\text{vdw}} + \mathbf{F}_i^{\text{cav}}\right), \tag{4.2.26}$$

where $\mathbf{r}_i$ is the absolute center position vector of the atom $a_i \in \mathbb{A}$ obtained from (4.2.9) and (4.2.10) in Section 4.2.1 (with different index notation), and $\mathbb{L}_j \subset \mathbb{A}$ is the subset of atoms that belong to the j^{th} link along the chain. Note that the origin of the absolute coordinate system (arbitrarily picked the same as the N-terminus anchor of the chain) is selected as the torque center for all links.

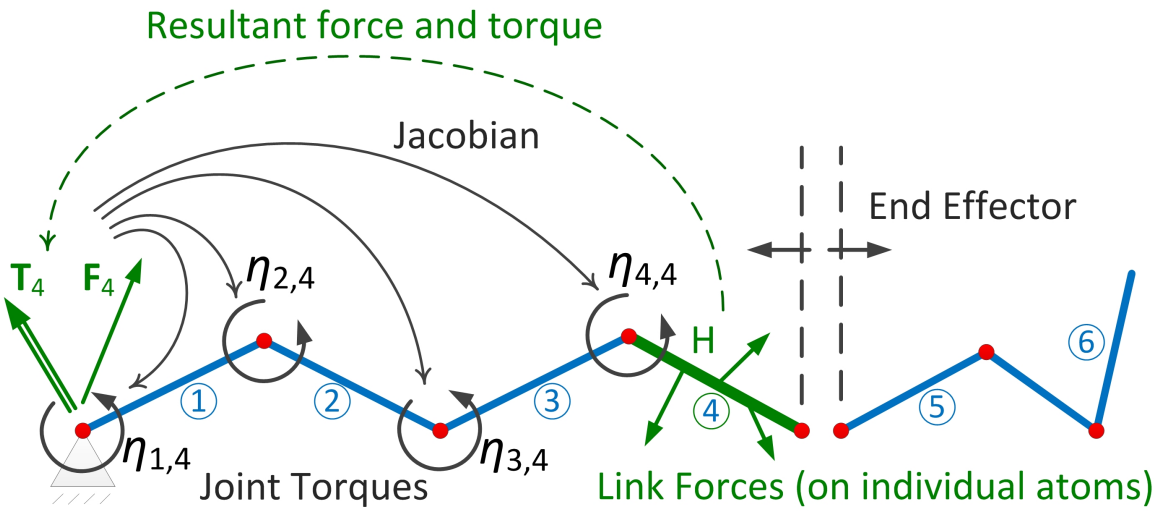


FIGURE 4.2.2: The forces on each link are converted into an equivalent set of joint torques on the preceding joints in the chain.

Equivalent Joint Torques. Since the revolute joints are assumed to be frictionless, the action of the link forces and torques can be replaced by an equivalent set of torques acting on the joints [85], as shown in Fig. 4.2.2. For a given rigid link, one can trace a unique serial chain of h successive links ($1 \leq h \leq l$) starting from the N-terminus and ending at the link under consideration, which is equivalent to a path along the graph of the open linkage. The contribution of the force $\mathbf{F}_h^{\text{link}}$ and the torque $\mathbf{T}_h^{\text{link}}$ applied to the end-effector link (i.e., the h^{th} link along the serial chain) to the joint torque at the k^{th} joint along the chain preceding the end-effector, denoted as $\eta_{k,h}$ ($1 \leq k \leq h, 1 \leq h \leq l$), can be computed using the conventional manipulator Jacobian matrix $[J]$ [85]:

$$[\boldsymbol{\eta}_h] = [J]^{\mathrm{T}} \begin{bmatrix} \mathbf{T}_h^{\text{link}} \\ \mathbf{F}_h^{\text{link}} \end{bmatrix}, \tag{4.2.27}$$

where $[\boldsymbol{\eta}_h] = [\eta_{1,h}, \eta_{2,h}, \cdots, \eta_{h,h}]^{\mathrm{T}}$ represents an $h \times 1$ array of joint torques that the end-effector force and torque will induce on the different joints preceding the end-

effector along the serial chain. $[J]^{\mathrm{T}}$ is the transpose of the $6 \times h$ Jacobian matrix $[J] = [\mathbf{J}_1, \mathbf{J}_2, \cdots, \mathbf{J}_h]$ for a given configuration of the chain [85]. The k^{th} column of the Jacobian associated with the k^{th} revolute joint is given by

$$[\mathbf{J}_k] = \begin{bmatrix} \mathbf{u}_k \\ \mathbf{u}_k \times (\mathbf{p}_h - \mathbf{p}_k) \end{bmatrix}, \tag{4.2.28}$$

where $\mathbf{u}_k$ is the unit vector along the k^{th} joint and $(\mathbf{p}_h - \mathbf{p}_k)$ is a vector connecting a point $\mathbf{p}_k$ along the k^{th} joint's axis to the point $\mathbf{p}_h$ where the end-effector force $\mathbf{F}_h$ applies—i.e., the atom positions obtained from (4.2.9) and (4.2.10). This process is repeated for all links to calculate the contribution of each link on the joints preceding that link in the serial chain. The total torque for each joint is obtained from a summation of these contributions [85]:

$$\tau_k = [\mathbf{J}_k]^{\mathrm{T}} \sum_{h=k}^{l} \begin{bmatrix} \mathbf{T}_h^{\mathrm{link}} \\ \mathbf{F}_h^{\mathrm{link}} \end{bmatrix}, \tag{4.2.29}$$

where the indexing $h = k, k+1, \cdots, l$ of the links is ordered along the main chain from amino-terminus to carboxyl-terminus, and can branch along the side chains. The range of the summation in (4.2.29) implies that each joint torque τ_k $(1 \leq k \leq l)$ depends on the forces $\mathbf{F}_h^{\mathrm{link}}$ and torques $\mathbf{T}_h^{\mathrm{link}}$ on the succeeding links $(k \leq h \leq l)$.

Kinetostatic Simulation. Making use of the assumption in [85] that the inertia forces at the atomic scale have negligible effects on the dynamics of folding compared to interatomic forces, the successive kinetostatic fold compliance relates the joint

torques to the changes in the dihedral angles as follows:

$$\Delta\theta_j = \kappa \frac{\tau_j}{|\tau_{max}|}, \tag{4.2.30}$$

where $\Delta\theta_j$ and τ_j are the angular change and the joint torque of the j^{th} revolute joint ($1 \leq j \leq l$), respectively. $|\tau_{max}|$ is the maximum joint torque throughout the entire chain, used to normalize the torques to the interval $[0, 1]$, and the coefficient κ is chosen small enough to avoid large changes in the angles, and to achieve numerical stability. One can notice that the conformational change computed in (4.2.30) is analogous to taking a step along the steepest-descent direction of the free energy gradient in the conformation landscape.

The computed changes in the joint angles are applied to the kinematic chain, and the entire process is repeated on the updated conformation until a convergence criteria is reached, as described in more detail is Section 4.4.

It is worthwhile noting that once the chain conformation (i.e., optimization variables) is modeled as in Section 4.2.1 and the energy-field (i.e., objective function) is formulated as in Section 4.2.2, the search for local or global minima of the free energy in (4.2.11) can be undertaken using a variety of classical (e.g., conjugate-gradients) and stochastic (e.g., genetic algorithm) optimization methods. Since the focus of this article is mainly on force-field modeling and computing, we skip a detailed treatment of the search phase.

4.3 Algorithms

This section presents efficient algorithms and data structures to speed up the force field computation during kinetostatic iterations. To leverage the proximity information between the atoms, we use a 3D hash table data structure based on a uniform spatial grid in Section 4.3.2. To classify the interatomic interaction types based on chain topology to compute the electrostatic and van der Waals effects, we use a tree-based data structure in Section 4.3.3 that replaces the adjacency matrix used in Protofold I [83–85]. To compute the solvation effects, we develop an approximate (yet adequately accurate) surface enumeration technique in 4.3.4, efficient CPU- and GPU-parallel implementations of which will be detailed in Section 4.4. Finally, we compute joint torques by aggregating contributions of different links (traversed along different paths in the linkage graph) on the joints along the chain, using the well-known 'prefix computation' in Section 4.3.5 which can also be implemented efficiently in parallel [37]. We show that the computational complexity of all steps is decreased from $O(n^2)$ in Protofold I [83–85] to expected $O(n)$ in Protofold II for a protein chain with a total of n atoms.

4.3.1 Rigid Transformations

At every snapshot $t \geq 0$ of the KCM, the protein conformation is described by a set of dihedral angles $\theta^t_{j,k}$ defined in (4.2.1) through (4.2.3).

- At the first iteration ($t = 0$), all angles are initialized as $\theta^0_{j,k} = 0$ (ZP conformation).
- At subsequent iterations ($t \geq 1$), for $1 \leq j \leq 2m$ (where m is the number

of AA residues) and $0 \leq k \leq l_i$ (where $0 \leq l_i \leq 4$ is the number of side chain links of the residue AA_i), the angles are obtained as $\theta^t_{j,k} = \theta^{t-1}_{j,k} + \Delta\theta^{t-1}_{j,k}$, where the increment $\Delta\theta^{t-1}_{j,k}$ is computed using (4.2.30) from the previous step's configuration and joint torques.

Once the dihedral angles are known, the transformation matrices $[M^t_{j,k}]$ are obtained from (4.2.7) and (4.2.8) using sequential matrix multiplication traversing the linkage tree from the anchored amino-terminus to the open carboxyl-terminus. Next, the unit vectors $\mathbf{u}^t_{j,k}$ and the body vectors $\mathbf{b}^t_{j,k}$ are computed from (4.2.5) and (4.2.6). Since the number of links is clearly less than the number of atoms, these computations take $O(n)$ time. The Cartesian coordinates of the individual atom center positions $\mathbf{r}_i \in \mathbb{R}^3$ $(1 \leq i \leq n)$ are obtained from the body vectors using (4.2.9) and (4.2.10), which also takes $O(n)$. In the following sections, we assume that both dihedral angles and atom center positions are known for the purpose of computing the next snapshot's energies, forces, and torques.

4.3.2 Proximity Queries

The brute-force approach for obtaining the proximity information at each snapshot is to check center distances against the cut-off distance for all possible pairs of atoms, which takes $O(n^2)$ time. Using this method, the approximate truncated formulae given in (4.2.12), (4.2.15), and (4.2.20) would take the same asymptotic time as the exact all-pairs formulae given in (4.2.13), (4.2.16), and (4.2.23), respectively, which is $O(n^2)$. Geometric hashing provides a simple solution to speed up proximity queries.

3D Hash Table. We use a 3D hash table data structure based on a uniform Cartesian grid, which bounds the current 3D structure of the protein and arranges the atom indices into groups based on their center positions. Each 3D grid cell is associated with a so-called 'bucket' that stores the indices of the atoms whose centers are located inside the cell in a linked-list. The grid dimensions are set dynamically to adapt to the shape of the protein's bounding box at the current snapshot.

The grid cells are chosen to be cubic, i.e., with equal edge length s_c along all 3 Cartesian axes. Given the min/max corner coordinates of the bounding box of the atom centers $\mathbf{r}_{\min}, \mathbf{r}_{\max} \in \mathbb{R}^3$—which can be obtained in $O(n)$ by scanning through the n atom center coordinates—we choose s_c in such a way that it results in $\lceil \alpha n \rceil$ grid cells/buckets, where $\alpha > 0$ is an arbitrary constant. More precisely, we choose $s_c = [v_{\mathrm{BB}}(\mathbb{A})/(\alpha n)]^{\frac{1}{3}}$ where $v_{\mathrm{BB}}(\mathbb{A})$ is the protein bounding box volume. The dimensions of the grid bounding box are then chosen as $\lceil (\mathbf{r}_{\max} - \mathbf{r}_{\min})/s_c \rceil s_c$ (slightly larger than the dimensions of the protein bounding box $\mathbf{r}_{\max} - \mathbf{r}_{\min}$), where the operator $\lceil \cdot \rceil$ is applied componentwise along the 3 Cartesian axes. Before we proceed with presenting the complexity analysis, we make the following assumptions:

Assumption 1. Due to the extremely strong repulsive van der Waals forces, the atoms that are not covalently bonded cannot penetrate into each other, and those covalently bonded intersect over a small volume. Given any arbitrary subset of atoms $\mathbb{A}' \subseteq \mathbb{A}$ with $R_{\min} = \min_{a_i \in \mathbb{A}'} R_i$ and $R_{\max} = \max_{a_i \in \mathbb{A}'} R_i$, let the maximum penetration volume between any pair of covalently bonded atoms $a_i, a_j \in \mathbb{A}'$ be upper-bounded by $\epsilon \min\{v_i, v_j\}$ where $v_i = \frac{4\pi}{3} R_i^3$ is the volume of the atom a_i, and $0 \leq \epsilon < \frac{1}{4}$ is a small number. Since each atom makes at most 4 covalent bonds, the unpenetrated volume for the atom a_i is lower-bounded by $(1 - 4\epsilon)v_i$, hence it is safe to

assume that $(1-4\epsilon) > 0$. Then the volume $v(\mathbb{A}')$ occupied by the union of all atoms in $\mathbb{A}'$ is bounded as $\frac{4\pi}{3}(1-4\epsilon)|\mathbb{A}'|R_{\min}^3 \leq v(\mathbb{A}') \leq \frac{4\pi}{3}|\mathbb{A}'|R_{\max}^3$. Consequently, there exists an 'average' radius $\bar{R}(\mathbb{A}')$ bounded as $(1-4\epsilon)^{\frac{1}{3}}R_{\min} \leq \bar{R}(\mathbb{A}') \leq R_{\max}$, such that $v(\mathbb{A}') = \frac{4\pi}{3}|\mathbb{A}'|\bar{R}^3(\mathbb{A}')$, where typically $\bar{R} \sim 1\mathring{A}$.

Assumption 2. It is also reasonable to assume that if the protein is either in an extended conformation aligned with one of the Cartesian axes (which is the case near the ZP conformation) or in a globular conformation (which is the case for most water-soluble proteins at their folded conformation), the empty space inside the bounding box is not extremely larger than the space occupied by the protein atoms, i.e., $v_{\text{BB}}(\mathbb{A})/v(\mathbb{A}) = O(1)$. Supported by experimentation, we assume this to be the case in the intermediate conformations as well, to simplify the analysis. However, there are possible conformations (e.g., extended along a diagonal direction in the axis-aligned bounding box) that would violate this assumption and result in slightly larger running times than predicted here, in spite of the low probability.

Letting $\mathbb{A}' := \mathbb{A}$ (hence $|\mathbb{A}'| = n$) in Assumption 1, and noting from the definition that $v_{\text{BB}}(\mathbb{A}) = (\alpha n)s_c^3$, from Assumption 2 it follows that $\frac{3\alpha}{4\pi}(s_c/\bar{R})^3 = O(1)$. Therefore, if we choose the grid cell size $s_c \sim 1\mathring{A}$ then $\alpha = O(1)$ and the number of buckets will be $\lceil \alpha n \rceil = O(n)$.

Table Construction. At each snapshot of the simulation, the algorithm scans through all the atoms with updated positions, and the deterministic hash function simply maps the atom center positions $\mathbf{r}_i \in \mathbb{R}^3$ that lie inside a grid cell into the corresponding 3D array of buckets. The 3D index $\mathbf{k} \in \mathbb{Z}^3 \cap [0, +\infty]^3$ of the bucket to which a given atom $a_i \in \mathbb{A}$ belongs is determined in $O(1)$ time as $\mathbf{k} = \lfloor (\mathbf{r}_i -$

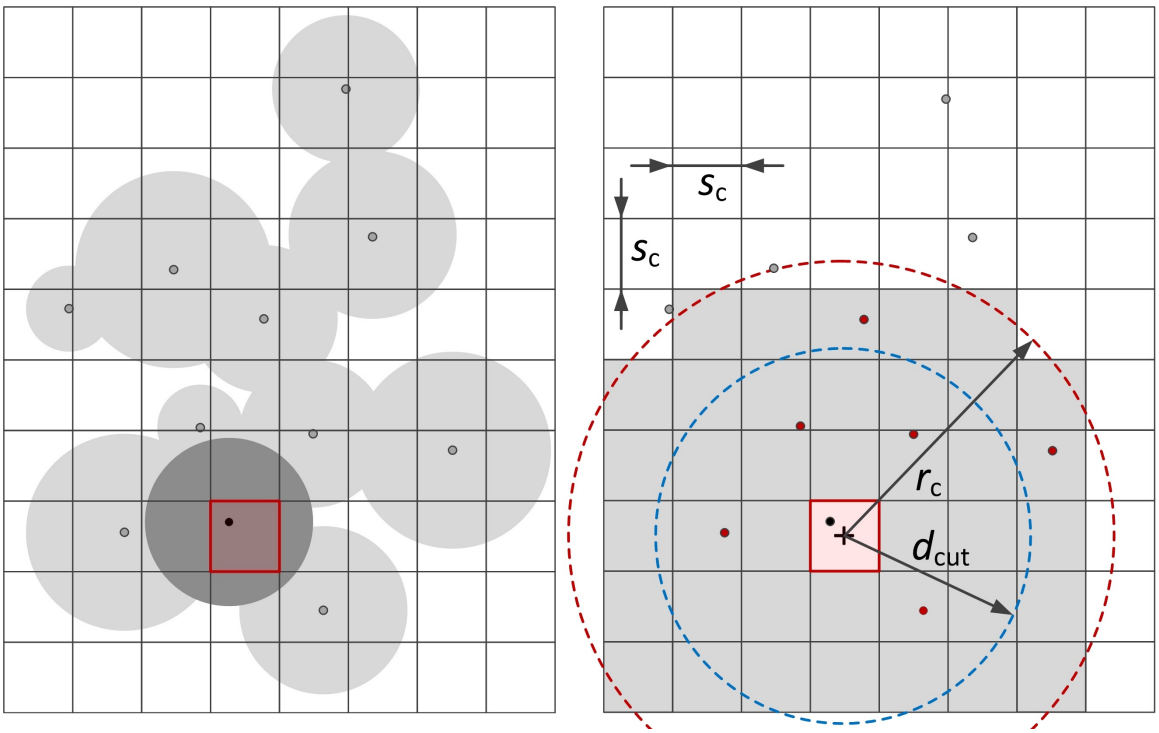


FIGURE 4.3.1: Atom centers are hashed into a 3D grid and the neighbors are selected within a cut-off distance.

$\mathbf{r}_{\min})/s_c\rfloor$, where the operator $\lfloor\cdot\rfloor$ is applied componentwise along the 3 Cartesian axes. Therefore, scanning through the atoms and constructing the 3D grid data structure is expected to $O(n)$ time and to requires $O(n)$ space.

Neighbor Queries. Once the atoms are arranged into the buckets, the algorithm iterates through the grid cells and scans through the linked-lists within the buckets. For each atom a_i in a given bucket associated with the grid cell index $\mathbf{k}$, the set of 'neighbor atoms' defined as

$$\mathbb{A}_i = \{a_j \in \mathbb{A} \mid \|\mathbf{r}_i - \mathbf{r}_j\|_2 \leq d_{\text{cut}}\}, \quad d_{\text{cut}} \in \{d_{\text{cut}}^{\text{elec}}, d_{\text{cut}}^{\text{vdw}}, d_{\text{cut}}^{\text{cav}}\} \tag{4.3.1}$$

can be identified rapidly for a given cut-off distance d_{cut} associated with any of the energetic interactions explained in Section 4.2.2. As illustrated in Fig. 4.3.1, a spherical region of radius $r_c = d_{\text{cut}} + \sqrt{3}s_c$ is considered around the (center point of) each grid cell to look for the (center point of) 'neighbor cells', defined as the collection of cells which *completely* lie inside this spherical region. The cut-off distance d_{cut} is offset by the diagonal size of the cells $\sqrt{3}s_c$ which takes into account the worst-case difference of the distance between cell centers and the distance between atom centers. This guarantees that the set of all atoms inside this collection of covered cells (denoted as $\mathbb{A}_i'$) contains the set of all neighbor atoms, i.e., $\mathbb{A}_i' \supseteq \mathbb{A}_i$, where $\mathbb{A}_i$ is one of the neighbor sets $\mathbb{A}_i^{\text{elec}}$, $\mathbb{A}_i^{\text{vdw}}$, or $\mathbb{A}_i^{\text{cav}}$. Letting $\mathbb{A}' := \mathbb{A}_i'$ in Assumption 1, the volume occupied by this set of atoms is $v(\mathbb{A}_i') = \frac{4\pi}{3}|\mathbb{A}_i'|\bar{R}^3(\mathbb{A}_i')$. Noting that $\mathbb{A}_i'$ is contained inside the spherical region of radius r_c, $v(\mathbb{A}_i') \leq \frac{4\pi}{3}r_c^3$ hence $|\mathbb{A}_i'| \leq (r_c/\bar{R})^3 = [(d_{\text{cut}} + \sqrt{3}s_c)/\bar{R}]^3 = O(1)$, since $\bar{R}, s_c \sim 1\mathring{A}$ and $d_{\text{cut}} \sim 10\mathring{A}$. As a result, it is expected to take $O(1)$ time to scan through the atoms inside the collection

of neighbor cells, and $O(n)$ total time to traverse all pairs of atoms using the 3D hash table.

For parallel implementation purposes, we construct (and dynamically maintain) a 'neighborhood matrix' composed of an array of n $O(1)-$sized linked-lists (one list per atom), where the i^{th} list ($1 \leq i \leq n$) contains the indices of the neighbor atoms $\mathbb{A}'_i$. Constructing this data structure is expected to take $\sum_{i=1}^{n} |\mathbb{A}'_i| = \sum_{i=1}^{n} O(1) = O(n)$ time and space, and accessing each atom's neighbors is expected to take $|\mathbb{A}'_i| = O(1)$ time.

Energy and Force Computations. Once the pairs of neighbors are identified, computing their electrostatic and van der Waals forces can be done in $O(1)$ time per pair, using the analytical truncated equations given in (4.2.18) and (4.2.19), respectively. It is important to note that such interactions are *pairwise*, i.e., they depend on the relative positions of pairs of atoms $a_i \in \mathbb{A}$ and $a_j \in \mathbb{A}_i^{\text{elec}}$ or $\mathbb{A}_i^{\text{vdw}}$, and the presence of any third atom $a_k \in \mathbb{A}$ ($k \neq i, j$) does not affect the force exchanged between a_i and a_j. Therefore, the computation algorithm is straightforward: it iterates over all atoms for $1 \leq i \leq n$ (sequentially or in parallel) and for each atom, it computes $\mathbf{F}_i^{\text{elec}}(\mathbb{A}'_i)$ and $\mathbf{F}_i^{\text{vdw}}(\mathbb{A}'_i)$, by sequentially aggregating the contributions of the hashed neighbor atoms $a_j \in \mathbb{A}'_i$, using (4.2.18) and (4.2.19), respectively.

Unfortunately, this is *not* the case for solvation force computation using (4.2.23) or (4.2.24), which requires computing the gradients of the atomic SASA with respect to the coordinates of the set of neighbor atoms. The SASA variations in one atom $a_i \in \mathbb{A}$ with respect to an infinitesimal change in the position of another atom $a_j \in \mathbb{A}_i^{\text{cav}}$, can be affected by the presence of a third atom $a_k \in \mathbb{A}_i^{\text{cav}}$ ($k \neq i, j$), thus

cannot be obtained in a pairwise fashion. This is because the overlaps of the pairs of offset spheres are not mutually disjoint. A segment of the offset surface can be overlapped with more than one neighbor sphere simultaneously, thus displacing one of the overlapping spheres may or may not affect the SASA. We return to this subject in Section 4.3.4 where a surface sampling algorithm is proposed as a simple solution.

4.3.3 Bonds Tree/Graph

To decide the weight factors $w_{i,j}^{\text{elec}}$ in (4.2.18) and $w_{i,j}^{\text{vdw}}$ in (4.2.19), one needs to quickly identify the types of interactions based on the number of bonds between pairs of atoms as described in Section 4.2.2. An $n \times n$ look-up table was used in Protofold I [83–85] for all pairs of n atoms which required a preprocessing step with $O(n^2)$ time and space. This can be improved by constructing a tree/graph data structure that stores the combinatorial structure of the chain, in which the vertices are atoms and edges are the covalent bonds between them. By excluding a single edge from the loops associated with the rare aromatic groups in certain side chains (e.g., imidazole in His and indole in Trp), this graph can be converted to a tree whose root is arbitrarily chosen as the N atom of the amino-terminus. The interaction types are then identified by the shortest path lengths between atoms.

The algorithm starts from the root and visits all vertices using a standard tree traversal routine. For each vertex, it stores a pointer to its parent and the index of the corresponding atom's AA residue. During the force computation in each KCM iteration, the residue indices for a pair of atoms of interest are checked. If the atoms are farther than a residue apart (i.e., if AA indices are neither identical nor consecutive), the weights in (4.2.18) and (4.2.19) are simply set to 1 (i.e., 1-4 interactions or

beyond). Otherwise, the algorithm checks 1) if one atom is the parent of the other (i.e., 1-2 interaction); 2) if one atom is the grand parent or sibling of the other (i.e., 1-3 interaction); or 3) if one atom is the grand grand parent or sibling of parent of the other (1-4 interactions). This requires $O(n)$ time and space for preprocessing and $O(1)$ query time during the KCM iterations.

4.3.4 Surface Enumeration

As mentioned in Section 4.1.1, several attempts have been made to approximate the SASA and its derivative by a pairwise treatment of the overlaps, including the probabilistic methods [3,55,175,180], popular in many molecular simulation software such as CHARMM [21], GROMOS [168], and AMBER [38,174]. In an early attempt to add solvation effects to Protofold II, we used these pairwise approximate formulae, which made it possible to compute the solvation forces with running times comparable to those of the electrostatic and van der Waals force computations. However, a comparison with the exact method [132] showed that when the distribution of the atoms deviates from that assumed in the the probabilistic methods, prohibitively large errors can be introduced into the resultant effects. The exact method [132] takes $O(|\mathbb{A}'_i|)$ operations for computing the SASA and its gradient for $a_i \in \mathbb{A}$ by using the coordinates of all neighbor atoms $\mathbb{A}_i^{\text{cav}} \subseteq \mathbb{A}'_i$. This is asymptotically $O(1)$ time per atom (since we reasoned earlier that $|\mathbb{A}'_i| = O(1)$ under Assumption 1), but in practice it is not nearly as fast as using the pairwise formulae. Alternatively, we use an approximate method that relies on an enumeration of the surface area, in which the deviations from the exact results can be controlled to a desired precision in a trade-off with computation time.

Offset Sphere Sampling. For a given atom $a_i \in \mathbb{A}$ of van der Waals radius R_i, an offset sphere of radius $R_i^{\text{off}} = R_i + R_{\text{H}_2\text{O}}$ concentric with the atom sphere is considered. The atom's SASA is obtained by measuring the area A_i^{exp} of the portion of the offset surface that is not overlapped by the offset sphere of any neighbor atom $a_j \in \mathbb{A}_i^{\text{cav}}$ (hence exposed to the solvent). To approximate the SASA, one can generate a large but finite 'quasi-uniform' set of sample points denoted as Q_i $(1 \leq i \leq n)$ on the surface of the offset sphere of the atom $a_i \in \mathbb{A}$, by which we mean a sampling that allows approximating the exposed fraction of the surface by the ratio of the number of exposed sample points to the total number of sample points. In other words, if we let

$$Q_i^{\text{exp}} = \{\mathbf{q} \in Q_i \mid \nexists a_j \in \mathbb{A}_i' \ : \ \|\mathbf{q} - \mathbf{r}_j\|_2 \leq R_j^{\text{off}}\} \tag{4.3.2}$$

be the subset of the solvent-exposed sample points, i.e., the points that are outside the offset spheres of all neighbors, then $\lim_{|Q_i|\to\infty} |Q_i^{\text{exp}}|/|Q_i| = A_i^{\text{exp}}/A_i^{\text{off}}$. If we define the 'exposure ratio' as $f_i^{\text{exp}} = |Q_i^{\text{exp}}|/|Q_i|$, the SASA can be approximated as $A_i^{\text{exp}} \approx f_i^{\text{exp}} A_i^{\text{off}}$ where $A_i^{\text{off}} = 4\pi(R_i^{\text{off}})^2$, using a large enough sample size $|Q_i| \gg 1$.[8]

There are different ways to obtain a quasi-uniform deterministic sampling on a sphere with consistent incremental quality [185]. For example, one could use a triangular spherical meshing algorithm, which starts from an icosahedron approximation of the sphere and recursively creates successive triangular subdivisions projected back on the sphere. Alternatively, one could use a polar geodesic sampling algorithm, which starts from a set of orbits with uniform angular distribution and samples a

[8] An alternative approach is uniform random sampling, e.g., using the simple method in [114]. Random sampling is easier to implement in parallel since every sample point in Q_i would be independent from others, and results in $A_i^{\text{exp}} = [f_i^{\text{exp}}]A_i^{\text{off}}$ where $[f_i^{\text{exp}}]$ is the *expected* ratio of the exposed sample points in probabilistic terms. However, it requires much larger sample sizes to approach the expectation and to achieve adequate accuracy in practice.

number of points uniformly on each orbit proportional to the orbit's circumference. We take the latter approach whose details are presented in [15].

To improve the efficiency, one could always precompute the coordinates for a single sampling Q on a unit sphere centered at the origin, and map it into individual atoms with different offset sphere center positions $\mathbf{r}_i$ and radii R_i^{off} using the mapping $Q_i = T_i(Q)$ where $T_i(\mathbf{q}) = \mathbf{r}_i + R_i^{\text{off}}\mathbf{q}$ for $1 \leq i \leq n$ and $\mathbf{q} \in Q$. This implies an equal number of sample points $N = |Q|$ for all atoms, selected as $N = 4\pi(R_{\text{max}}^{\text{off}})^2/\delta A$ where $R_{\text{max}}^{\text{off}} = R_{\text{max}} + R_{\text{H}_2\text{O}}$ is the maximum offset sphere radius, and δA is the desired characteristic area element carried by each sample point. Hence the sampling takes $O(nN)$ operations for all atoms regardless of the sampling technique. For implementation purposes, we assign an arbitrary ordering to the sample points, letting $Q = \{\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_N\}$ and denoting transformed sample points as $\mathbf{q}_{i,k} = T_i(\mathbf{q}_k)$.

Energy and Force Approximations. Substituting for $A_i^{\text{exp}} \approx f_i^{\text{exp}} A_i^{\text{off}} = 4\pi f_i^{\text{exp}}(R_i^{\text{off}})^2$ into (4.2.20), the total solvation free energy can be computed directly from

$$G^{\text{cav}}(\mathbb{A}) \approx \sum_{a_j \in \mathbb{A}} 4\pi\gamma_j f_j^{\text{exp}}(R_j^{\text{off}})^2. \tag{4.3.3}$$

To obtain the solvation force on any atom $a_i \in \mathbb{A}$, the energy must be differentiated with respect to the atom's center coordinates $\mathbf{r}_i \in \mathbb{R}^3$, giving rise to (4.2.23). It is very important to note that an infinitesimal displacement of the atom a_i can change the SASA of a_i itself, as well as that of the neighbor atoms $a_j \in \mathbb{A}_i^{\text{cav}} \subseteq \mathbb{A}_i'$. However, we showed in Section 4.2.2 that the two effects are geometrically dependent, yielding the symmetric form in (4.2.24). From a computational perspective, (4.2.24) is preferred over (4.2.24), because

1. it eliminates the need for computing the gradient $\nabla_{\mathbf{r}_j} A_i^{\text{exp}}$ for the cases when $i = j$, hence decreasing the number of such computations from n^2 to $n(n-1)$; and

2. its symmetric form lends itself to a data-parallel implementation that is balanced between computation and data sharing tasks, as we show in Section 4.4.

This follows from the fact that for a given pair of indices $i \neq j$, the first term in the formula for $\mathbf{F}_i^{\text{cav}}$ is identical to the second term in the formula for $\mathbf{F}_j^{\text{cav}}$ (both given by (4.2.24)), cutting the number of required SASA gradient computations to $n(n-1)/2$. Substituting for A_i^{exp} and A_j^{exp} in (4.2.24), the solvation forces can be approximated as

$$\begin{aligned}\mathbf{F}_i^{\text{cav}}(\mathbb{A}_i') \approx 4\pi\gamma_i(R_i^{\text{off}})^2 \sum_{a_j \in \mathbb{A}_i'} \nabla_{\mathbf{r}_j} f_i^{\text{exp}} \\ - 4\pi \sum_{a_j \in \mathbb{A}_i'} \gamma_j (R_j^{\text{off}})^2 \nabla_{\mathbf{r}_i} f_j^{\text{exp}},\end{aligned} \tag{4.3.4}$$

where $\nabla_{\mathbf{r}_j} f_i^{\text{exp}}$ and $\nabla_{\mathbf{r}_i} f_j^{\text{exp}}$ can be approximated using the forward-difference method from finite variations of f_i^{exp} and f_j^{exp} with respect to the positions of the atoms a_i and a_j, respectively:

$$\nabla_{\mathbf{r}_j} f_i^{\text{exp}} \approx \sum_{s=1,2,3} \frac{f_{i,j,s}^{\text{exp}} - f_i^{\text{exp}}}{\delta r} \mathbf{e}_s, \tag{4.3.5}$$

where $\delta r > 0$ is the finite difference, $\mathbf{e}_s$ $(s = 1, 2, 3)$ are the unit vectors along the 3 Cartesian axes, and $f_{i,j,s}^{\text{exp}}$ are the exposure ratio of the atom $a_i \in \mathbb{A}$ after changing the position of the neighbor atom $a_j \in \mathbb{A}_i'$ from the current value $\mathbf{r}_j$ to a hypothetical variant $\mathbf{r}_j + \delta r \mathbf{e}_s$.

Enumeration Algorithm. In order to compute the exposure ratio and its finite-difference gradient, we use a binary enumeration function $B : \mathbb{A} \times Q \to \{0, 1\}$ ($1 \leq i \leq n$) such that $B(a_i, \mathbf{q}_k) = 1$ if the sample point $\mathbf{q}_{i,k} \in T_i(Q)$ on the offset sphere of the atom $a_i \in \mathbb{A}$ is overlapped by at least one neighbor offset sphere (i.e., if $\exists a_j \in \mathbb{A}'_i$ such that $\|T_i(\mathbf{q}_k) - \mathbf{r}_j\|_2 \leq R_j^{\text{off}}$) and $B(a_i, \mathbf{q}_k) = 0$ if the sample point is exposed to the solvent. The algorithm iterates over all atoms for $1 \leq i \leq n$ and all sample points for $1 \leq k \leq N$ (sequentially or in parallel). For each sample point, the indicator $B_{i,k} := B(a_i, \mathbf{q}_k)$ is initialized to 0, and each point is tested against the set of neighbors $\mathbb{A}'_i$, scanned sequentially. As soon as one overlapping neighbor is found, $b_{i,k}$ is set to 1 and there is no need to test the rest of the neighbors. The exposure ratio is then computed as

$$f_i^{\text{exp}} = 1 - \sum_{\mathbf{q}_k \in Q} \frac{B(a_i, \mathbf{q}_k)}{|Q|} = 1 - \sum_{k=1}^{N} \frac{B_{i,k}}{N}. \tag{4.3.6}$$

In the worst case, this takes $|\mathbb{A}'_i|N$ tests and N binary sums per atom where $N = |Q|$ is the sample size, which adds to $O(N)$ basic operations per atom (since we reasoned earlier that $|\mathbb{A}'_i| = O(1)$ under Assumption 1), and a total of $O(nN)$ time for all atoms.

For every sample point, the sequential inner loop of the algorithm can be repeated $3|\mathbb{A}'_i|$ times for computing the variations $f_{i,j,1}^{\text{exp}}$, $f_{i,j,2}^{\text{exp}}$, and $f_{i,j,3}^{\text{exp}}$ used in (4.3.5), after introducing the finite difference to the 3 Cartesian coordinates (one at a time) of each neighbor atom $a_j \in \mathbb{A}'_i$. This takes $3|\mathbb{A}'_i|^2 N$ more tests per atom, still asymptotically $O(N)$ but not fast enough in practice. There is a notably more efficient way to do the latter computation by ruling out the subset of sample points that cannot possibly contribute to $f_{i,j,s}^{\text{exp}} - f_i^{\text{exp}}$ ($s = 1, 2, 3$) in (4.3.5) during the first iteration

Algorithm 1: SASA enumeration algorithm for solvation free energy and force computation.

Input: $\mathbf{r}_i, R_i^{\text{off}}, \gamma_i, Q_i,$ and $\mathbb{A}'_i$ for all $a_i \in \mathbb{A}$ $(1 \le i \le n)$;
Output: G_i^{cav} and $\mathbf{F}_i^{\text{cav}}$ for all $a_i \in \mathbb{A}$ $(1 \le i \le n)$;

for $1 \le i \le n$ (seq. or in $\|$) **do**
  Step 1: Energy Computation:
  initialize $f_i^{\text{exp}} \leftarrow 1$;
  initialize $G_i^{\text{cav}} \leftarrow G_{i,0}^{\text{cav}} \leftarrow 4\pi\gamma_i (R_i^{\text{off}})^2$;
  for $1 \le k \le |Q_i|$ (seq. or in $\|$) **do**
    initialize $C_{i,k} \leftarrow 0$; $j_{i,k}^{\text{over}} \leftarrow -1$;
    for $j =$ *indices of atoms in* $\mathbb{A}'_i$ (seq.) **do**
      if $\|\mathbf{q}_{i,k} - \mathbf{r}_j\|_2 \le R_j^{\text{off}}$ **then**
        increment $C_{i,k} \leftarrow C_{i,k} + 1$;
        if $C_{i,k} = 1$ **then**
          //Save critical neighbor index:
          write $j_{i,k}^{\text{over}} \leftarrow j$;
          atomic read+modify+write
            $f_i^{\text{exp}} \leftarrow f_i^{\text{exp}} - 1/|Q_i|$;
            $G_i^{\text{cav}} \leftarrow G_i^{\text{cav}} - G_{i,0}^{\text{cav}}/|Q_i|$;
        else
          if $C_{i,k} \ge 2$ **then**
            write $j_{i,k}^{\text{over}} \leftarrow -1$;
            break;
  Step 2: Force Computation:
  initialize $f_{i,1}^{\text{exp}} \leftarrow f_{i,2}^{\text{exp}} \leftarrow f_{i,3}^{\text{exp}} \leftarrow f_i^{\text{exp}}$;
  initialize $F_{i,1}^{\text{cav}} \leftarrow F_{i,2}^{\text{cav}} \leftarrow F_{i,3}^{\text{cav}} \leftarrow 0$;
  Synchronize for all $1 \le i \le n$;
  for $1 \le k \le |Q_i|$ (seq. or in $\|$) **do**
    for $1 \le s \le 3$ (seq. or in $\|$) **do**
      if $C_{i,k} = 0$ **then**
        for $j =$ *indices of atoms in* $\mathbb{A}'_i$ (seq.) **do**
          if $\|\mathbf{q}_{i,k} - (\mathbf{r}_j + \delta r \mathbf{e}_s)\|_2 \le R_j^{\text{off}}$ **then**
            atomic read+modify+write
              $f_{i,j,s}^{\text{exp}} \leftarrow f_{i,j,s}^{\text{exp}} - 1/|Q_i|$;
              $F_{i,s}^{\text{cav}} \leftarrow F_{i,s}^{\text{cav}} - G_{i,0}^{\text{cav}}/(|Q_i|\delta r)$;
              $F_{j,s}^{\text{cav}} \leftarrow F_{i,s}^{\text{cav}} + G_{i,0}^{\text{cav}}/(|Q_i|\delta r)$;†
            break;
      else
        if $C_{i,k} = 1$ **then**
          write $j \leftarrow j_{i,k}^{\text{over}}$; //note: $j_{i,k}^{\text{over}} \ne -1$
          if $\|\mathbf{q}_{i,k} - (\mathbf{r}_j + \delta r \mathbf{e}_s)\|_2 > R_j^{\text{off}}$ **then**
            atomic read+modify+write
              $f_{i,j,s}^{\text{exp}} \leftarrow f_{i,j,s}^{\text{exp}} + 1/|Q_i|$;
              $F_{i,s}^{\text{cav}} \leftarrow F_{i,s}^{\text{cav}} + G_{i,0}^{\text{cav}}/(|Q_i|\delta r)$;
              $F_{j,s}^{\text{cav}} \leftarrow F_{i,s}^{\text{cav}} - G_{i,0}^{\text{cav}}/(|Q_i|\delta r)$;†
            break;
  write $\mathbf{F}_i^{\text{cav}} \leftarrow (F_{i,1}^{\text{cav}}, F_{i,2}^{\text{cav}}, F_{i,3}^{\text{cav}})$;

//† The instructions that require architecture-specific mutex.

when computing $B_{i,k}$ indicators. In particular, if a sample point is overlapped by more than one neighbor, displacing any neighbor does *not* affect its exposure state (from overlapped to exposed or vice versa), hence it does not contribute to $f_{i,j,s}^{\text{exp}} - f_i^{\text{exp}}$. To leverage this property, we expand the binary definition of the state function to $C : \mathbb{A} \times Q \to \mathbb{Z} \cap [0, \infty)$ such that $C(a_i, \mathbf{q}_k)$ counts the actual number of neighbors $a_j \in \mathbb{A}_i'$ that overlap the sample point $\mathbf{q}_{i,k} \in T_i(Q)$. Three different states for a sample point are observed in terms of the changes in $C_{i,k} := C(a_i, \mathbf{q}_k)$:

1. 'Not overlapped' or 'exposed' ($C_{i,k} = 0$). In this case, displacing any neighbor either keeps the state at $C_{i,k} = 0$ or changes it to $C_{i,k} = 1$, where the latter case affects the contribution to SASA. Hence the inner loop needs to be repeated for all neighbors (i.e., for $3|\mathbb{A}_i'|$ times).

2. 'Critically overlapped' ($C_{i,k} = 1$). In this case, the only neighbor whose displacement may change the sample point's state to $C_{i,k} = 0$ is the one that originally overlapped it, and displacing any other neighbor either keeps the state at $C_{i,k} = 1$ or changes it to $C_{i,k} = 2$, both of which correspond to overlapped states that does *not* affect the contribution to SASA. Hence the inner loop is repeated only 3 times after displacing that critical neighbor along the 3 Cartesian axes.

3. 'Multiply overlapped' ($C_{i,k} \geq 2$). In this case, displacing any neighbor either keeps the state at $C_{i,k} \geq 2$ or changes it to $C_{i,k} = 1$, both of which correspond to overlapped states that does *not* affect the contribution to SASA. Hence the inner loop need not be repeated at all.

Therefore, the only changes that contribute a nonzero value to $f_{i,j,s}^{\text{exp}} - f_i^{\text{exp}}$ ($s =$

$1, 2, 3$) are those from exposed ($C_{i,k} = 0$) to critically overlapped ($C_{i,k} = 1$) and vice versa, thus a significant amount of computation time can be saved by early detection of the rest. An atom $a_j \in \mathbb{A}'_i$ is called a 'critical neighbor' of the atom $a_i \in \mathbb{A}$ with respect to a sample point $\mathbf{q}_{i,k} \in Q_i$ along a particular direction $\mathbf{e}_s$ ($s = 1, 2, 3$), if a finite displacement $\delta r \mathbf{e}_s$ results in such a change. As a direct consequence of geometry, if a_j is a critical neighbor of a_i along $+\mathbf{e}_s$, then a_i is also a critical neighbor of a_j along $-\mathbf{e}_s$, both with respect to the same sample point. Therefore, a pair of neighbor atoms $a_i, a_j \in \mathbb{A}$ exchange a solvation force $\pm\delta\mathbf{F}^{\text{cav}} = \pm 4\pi\gamma_i (R_i^{\text{off}})^2/(N\delta r)\mathbf{e}_s$ due to their overlap at the sample point $\mathbf{q}_{i,k}$ if and only if they are critical neighbors with respect to $\mathbf{e}_s$. The improved algorithm (based on the integer-valued $C_{i,k}$) is different from the original (based on the binary-valued $B_{i,k}$) in that the first iteration of the sequential inner loop for computing $C_{i,k}$ terminates after the *second* (rather than the *first*) overlap is encountered, because all $C_{i,k} \geq 2$ have equivalent implications according to the above rules.[9] During this step, the value of f_i^{exp} is initialized to 1 for each atom, and every time a sample point with $C_{i,k} = 1$ or 2 is discovered, f_i^{exp} is decremented by $1/N$. The next 3 repetitions of the inner loop per neighbor atom displacement depend on the aforementioned rules based on the value of $C_{i,k}$. The 3 variants of the exposure ratio $f_{i,j,s}^{\text{exp}}$ ($s = 1, 2, 3$) are initialized to f_i^{exp} for each atom with respect to displacements in all of its neighbors. Every time a sample point with $C_{i,k} = 0$ or 1 is encountered, the inner loop is repeated with displaced neighbor coordinates to discover the critical neighbors, each adding $\pm 1/N$ to $f_{i,j,s}^{\text{exp}}$.

Significant speed-ups are achieved in terms of the average time, a rigorous analysis of which is not possible without assumptions on the spatial distribution of atoms.

[9] Hence one could redefine to $C : \mathbb{A} \times Q \to \{0, 1, \text{“2 or more”}\}$ to implement the same trick with only 3 distinct flags, as in Algorithm 1.

However, the worst case time complexity is still $O(nN)$ for the sequential algorithm. One could easily parallelize the algorithm at the outer loops over the atoms and sample points, while the inner loops over the neighbor atoms is best implemented sequentially. On a simple CRCW PRAM machine with common conflict resolution (briefly introduced in Appendix C.1), the parallel running time of $O(nN/P)$ can be achieved in theory using P processors (i.e., linear speed-up), which leads to $O(n)$ if we have $P = O(N)$ processors at our disposal—not far from reality when using GPUs. However, there are more complications to the machine architecture in practice, as will be addressed in Section 4.4.

The complete process is described in pseudo-code in Algorithm 1. The instructions marked by a dagger (†) modify variables that belong to different atoms iterated in parallel by the outer-most loop, hence require a mutex with nuances that depend on the architecture as described in Section 4.4.

4.3.5 Prefix Sum Calls

There are multiple references in Protofold II to the generic prefix sum routine—explained in Appendix B, which can be performed using optimal sequential and parallel algorithms in linear number of steps—that emerge naturally as a consequence of the linear topology of the polypeptide backbone:

1. Computing link transformations from successive matrix multiplications in (4.2.7) and (4.2.8), in which the domain is $\Sigma := \mathrm{SO}(3)$ (represented by 3×3 rotation matrices) and the operator $\oplus$ is the matrix multiplication.

2. Computing atom center coordinates from successive vector summations in (4.2.9)

and (4.2.10), in which the domain is $\Sigma := \mathbb{R}^3$ (represented by 3×1 column matrices) and the operator $\oplus$ is the vector summation.

3. Computing joint torques from successive superposition of the contributions of each link on the preceding joints in the chain using (4.2.27) and (4.2.29), in which the domain is $\Sigma := \mathbb{R}^6$ (represented by 6×1 column matrices) and the operator $\oplus$ is the inner product.

The first item clearly takes $O(n)$ steps, while the latter two take $O(l) = O(m)$ steps (which is also $O(n)$). To explain the last item further, let $[J_k]$ be the k^{th} column of the Jacobian matrix $[J]$ and $[\mathbf{P}_h] := [\mathbf{T}_h^{\text{link}}\ \mathbf{F}_h^{\text{link}}]^{\text{T}}$ be the so-called generalized force on the right-hand side of (4.2.27) on the h^{th} link along the chain, both of which are 6×1 column matrices. The contribution of $\mathbf{P}_h$ on the k^{th} joint is obtained as the inner product of the two matrices $\eta_{k,h} = [\mathbf{J}_k]^{\text{T}}[\mathbf{P}_h]$ arranged into the following matrix:

$$[\eta] = \begin{bmatrix} \mathbf{J}_1^{\text{T}}\mathbf{P}_1 & \mathbf{J}_1^{\text{T}}\mathbf{P}_2 & \cdots & \mathbf{J}_1^{\text{T}}\mathbf{P}_l \\ 0 & \mathbf{J}_2^{\text{T}}\mathbf{P}_2 & \cdots & \mathbf{J}_2^{\text{T}}\mathbf{P}_l \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{J}_l^{\text{T}}\mathbf{P}_l \end{bmatrix}, \tag{4.3.7}$$

where $[\eta]$ is an $l \times l$ upper-triangular matrix made of the torque contributions $\eta_{k,h}$, whose h^{th} column's upper nonzero elements form the $h \times 1$ column matrix $[\boldsymbol{\eta}_h]$ introduced in (4.2.27). Note that each link only affects the preceding joints in the chain, hence $\eta_{k,h} = 0$ for all $h \leq k - 1$. The total torque joints $\tau_k (1 \leq k \leq l)$ can be obtained as a summation over the rows of the above matrix via (4.2.29). In Protofold I [83–85] this was accomplished by scanning through the terms along the columns in (4.3.7), which took $l(l+1)/2 = O(l^2)$ operations. In Protofold II we perform row

scanning of the matrix, starting from the bottom row and moving upwards. More specifically, by factoring out the Jacobian terms $[\mathbf{J}_k^{\mathrm{T}}]$ in each row of (4.3.7) and aggregating the generalized forces into $[\mathbf{P}_k^{\mathrm{agg}}] = \sum_{h=k}^{l}[\mathbf{P}_h]$, (4.2.29) yields $\tau_k = [\mathbf{J}_k]^{\mathrm{T}}[\mathbf{P}_k^{\mathrm{agg}}]$ as the sum of each row. Then $[\mathbf{P}_k^{\mathrm{agg}}]$ can be obtained in $O(1)$ time from $[\mathbf{P}_{k+1}^{\mathrm{agg}}]$ as $[\mathbf{P}_k^{\mathrm{agg}}] = [\mathbf{P}_k] + [\mathbf{P}_{k+1}^{\mathrm{agg}}]$, which leads to a total of $O(l)$ prefix computation steps.

4.4 Implementation

Figure 4.3.2 is a schematic of the Protofold II architecture elaborated in Section 4.4.1. The remainder of this section is dedicated to the parallel the implementation of Algorithm 1 on the CPU in Section 4.4.2 and on the GPU in Section 4.4.3, which are identified by the alternative shaded modules in Fig. 4.3.2.

4.4.1 Protofold II Architecture

Unlike Protofold I [83–85] that was programmed in Matlab®, Protofold II is reprogrammed with a new architecture in C++. The CPU- and GPU-parallel algorithms are implemented as external modules and linked to the main application thread as dynamic link libraries (DLL), which can be integrated into other folding packages.

As depicted in Fig. 4.3.2, a typical KCM simulation in Protofold II can be summarized into the following steps:

1. **Input:** The user specifies a primary structure (i.e., AA sequence information) to the interface.

2. **Preprocessing:** The program constructs the AA chain using the structural assumptions given in Section 4.2.1 to arrange the atoms into the consecutive

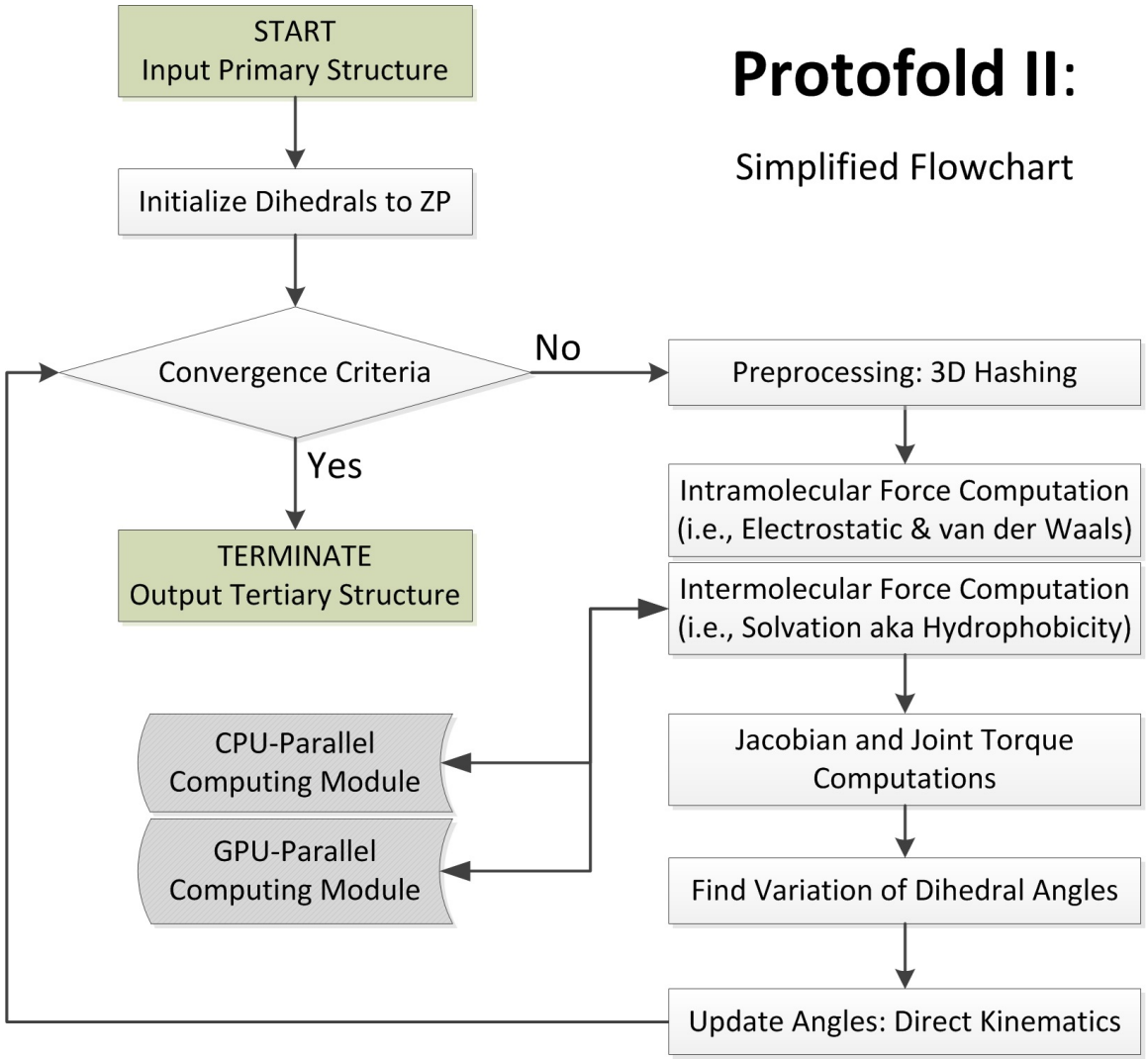


FIGURE 4.3.2: The main process flowchart of Protofold II

peptide planes. The double-bond angles are all set to the fixed values of $\omega_i = 0°$ (cis) or $-180°$ (trans) and the body vectors are assigned with the values given in Table 4.2.1.

3. **Initialization:** The main chain dihedral angles are initialized as $\phi_i^0 = \psi_i^0 = -180°$ and the side chain dihedral angles are initialized to rotameric default values $\chi_{i,k}^0$ [86] for $1 \leq i \leq n, 1 \leq k \leq l_i \leq 4$—i.e., set all $\theta_{j,k}^0 = 0°$ in (4.2.1) through (4.2.3) referred to as ZP initial conditions.

4. **Forward Kinematics:** The conformation variables summarized in Table 4.2.2 are converted to the Cartesian coordinates of the individual atoms by using the sequence of rigid body transformations described in Section 4.2.1.

5. **Coordinate Hashing:** Using the 3D grid data structure presented in Section 4.3.2, the atom coordinates are arranged into buckets for fast neighborhood queries based on the cut-off distances.

6. **Force Computations:** The free energy- and force-fields are computed from the atom coordinates using the equations given in Section 4.2.2. This is where the CPU- or GPU-parallel modules are called for computing the solvation effects.

7. **Torque Computations:** The forces on the atoms are converted to joint torques using the Jacobian transformation described in Section 4.2.2.

8. **KCM Stepping:** The kinetostatic effect of the joint torques are computed using the simple steepest-descent stepping explained in Section 4.2.3.

9. **Termination:** If the convergence criteria is met, the program terminates; otherwise it repeats the steps 4 through 8 above.

10. **Output:** The intermediate (every several frames) and final conformations in PDB format, the variations of the dihedral angles and free energy terms, and the performance measures (e.g., running times of different steps) are exported by the program.

These steps characterize the process of arriving from sequence configuration (i.e., primary structure) to stable 3D conformation (i.e., tertiary structure) without any additional assumptions. Although this is the ultimate goal of protein folding, it is rather ambitious to obtain results that are consistent with experimental observations except in the case of relatively short chains; e.g., folding simulation of $\alpha-$helix coiling described in section 4.5.1. This is due to a variety of reasons ranging from the sensitivity of the folding pathway to the physical parameters (e.g., adjusted coefficients in the empirical force-field equations) to the sensitivity of the spatial structure of long chains to simplifying geometric assumptions (e.g., the exact planarity of the peptide planes).

Additional Functionalities. In order to enable addressing certain computer-aided structural studies on real proteins effectively in spite of the aforementioned difficulties, we found it imperative to include the following additional functionalities in Protofold II:

- The user has the option to 1) specify only sequence data, from which the 'canonical' peptide plane geometry (i.e., assuming exact planarity $\omega_i = 0^\circ$ (cis) or -180° (trans) and average lengths in Table 4.2.2); or 2) import the protein structure as a PDB file and retain the peptide group geometry as-read when constructing the rigid links.

- The user has the option to limit the mobility of the linkage by fixing as many dihedral angles as desired. This enables folding studies at multiple levels and different scales. For example, it is possible to group collections of AAs (e.g., secondary elements, motifs, domains, etc.) into presumed rigid bodies and limit the DOF to deformations at the loops connecting them.

- In addition to the ZP initial conditions, the user may choose to use other initial conditions, including but not limited to completely random initial conditions or the native conformation perturbed by arbitrary (deterministic or randomized) changes to certain dihedral angles.

- When importing PDB files, the program eliminates water molecules—since their effect is implicitly incorporated by the solvation energies—but retains other heteroatoms (e.g., metal ions, co-factors, substrates, etc.) and includes them among chain atoms when computing the force-field. This is crucial since the proper folding of many proteins is dependent on these agents.

In addition to the above features, the following need to be included in future versions:

- The program currently supports monomeric protein folding in its simplest topology. It is desirable to enable multimeric protein folding by maintaining multiple chains bound together (i.e., quaternary structure) and more complex topologies induced by other effects (e.g., disulfide bonds, hydrogen bonds, lipidation, etc.)

- the simplistic steepest-descent search process presented in Section 4.2.3 has not evolved much since Protofold I [83–85]. Our numerical experiments suggest that better optimization algorithms such as a hybrid Monte Carlo sampling combined

with steepest-descent or conjugate-gradients KCM[10] could be more effective in avoiding local minima and enable faster convergence to the global minimum.

Parallel Implementations. As demonstrated in Section 4.3.4, the solvation energy and force computations using (4.2.24) and Algorithm 1 are the most time-consuming steps of each KCM iteration, mainly due to the large number of sample points $|Q| = N \gg 1$ required to enumerate the offset sphere of each atom for an adequate approximation of SASA and its gradient. To benefit from the single-instruction multiple-data (SIMD) characteristic of Algorithm 1, the variables pertaining to different atoms are assigned to different processors. The two terms on the right-hand side of (4.2.24) are computed concurrently by different processors assigned to $a_i, a_j \in \mathbb{A}$ and broadcasted to each other to minimize the computational work. An immediate consequence is an additional communication overhead and possible network contention due to concurrent write attempts. Such a trade-off between computation and communication intensities is a common characteristic of parallel algorithms [103], and will be considered here for code optimization.

Here we focus on the implementation of the SIMD Algorithm 1 using two parallel computing models; namely,

- one that is designed for coarse-grained multiprocesser machines such as multi-core CPUs (Section 4.4.2); and
- another that is designed for fine-grained multiprocesser machines such as many-core GPUs (Section 4.4.3).

[10]This module is already implemented into Protofold II but not tested yet, as the focus of this article is on the improved model and implementation of the force-field.

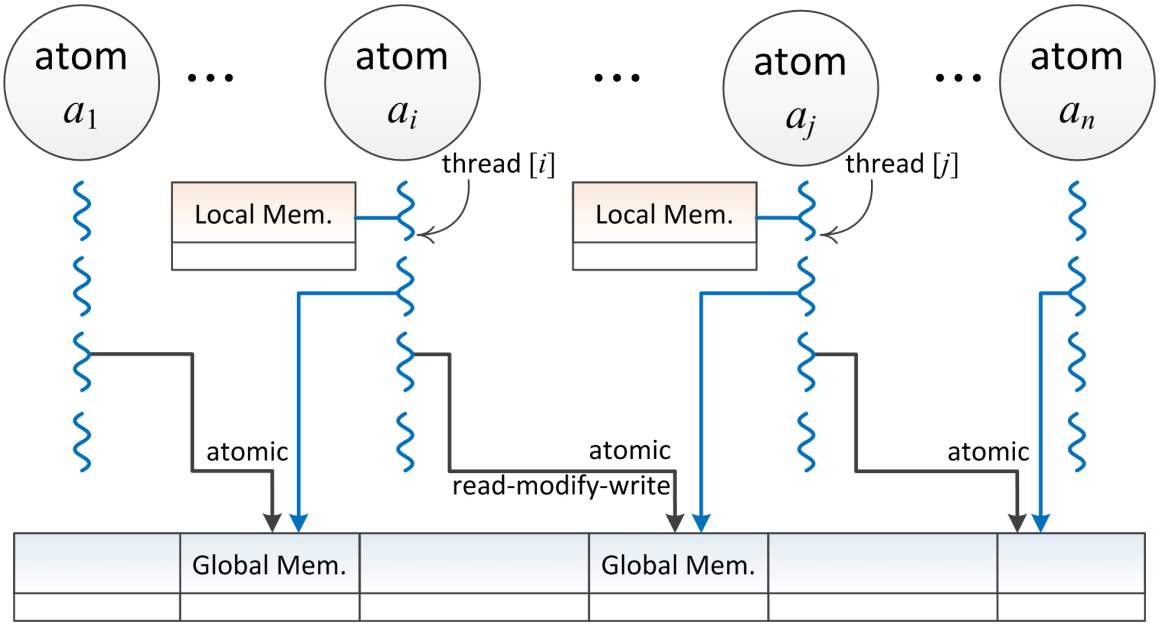


FIGURE 4.4.1: Thread execution model on the CPU.

4.4.2 CPU-Parallel Implementation

The first parallel implementation targets a coarse-grained shared-memory multiprocessor machine, i.e., one with a multi-core multi-thread CPU. Given n atoms n threads are generated, assigning one thread per atom a_i $(1 \leq i \leq n)$. The neighborhood information (i.e., the list of indices of all $a_j \in \mathbb{A}_i$ defined in (4.3.1) for each atom $a_i \in \mathbb{A}$) is constructed and saved in the *global* memory shared among all processors, hence can be accessed concurrently from different threads. Each thread stores the sample point coordinates, the SASA and its gradient, and the resulting solvation energy and force components in the *local* memory of the processor. The thread iterates sequentially over the sample points on the offset sphere, and a counter variable that keeps track of the number of overlapped sample points is initialized within the scope of the thread. For each sample point, the coordinates are computed and tested sequentially against all neighbors to obtain $C_{i,k}$ $(1 \leq i \leq n, 1 \leq k \leq |Q_i|)$.

Once the exposure states are obtained for the original configuration of the neighbors, the thread loops over all neighbors one more time to examine the effects of their displacement along the 3 coordinate axes one at a time. If certain criteria given in the previous section are met, the pair of force components $\pm\delta F^{\text{cav}} = \pm 4\pi\gamma_i (R_i^{\text{off}})^2/(N\delta r)$ need to be added to the total solvation forces of two neighbor atoms a_i and a_j along the proper coordinate axis, and in opposite directions. This results in two write operations per incidence, the first of which modifies $\mathbf{F}_i^{\text{cav}}$ of a_i, which is safely assigned to the current thread and occurs in the local memory without any concern related to communication between the threads. The second write operation, on the other hand, modifies $\mathbf{F}_j^{\text{cav}}$ of a_j, a variable assigned to a different thread. This requires communication between the two threads, and has to be implemented using atomic write operations into the global memory to guarantee mutual exclusion. Figure 4.4.1 shows the multi-threading scheme for the CPU-parallel algorithm. The algorithm is implemented using the OpenMP library. Although linear speed-up is expected in theory on an abstract CRCW PRAM, the actual speed-up is sublinear (as depicted in Section 4.5.2) in practice due to bus traffic, network contention, cache invalidations, and serialized operations.

CPU Optimization. The number of CPU cores is generally much smaller that the number of atoms ($p \ll n$). Nevertheless, it is good practice to generate more threads than the number of cores to maximize the performance by keeping the processors saturated at all times with computational work. Accessing global memory incurs latency at the incidence of a cache miss and multithreading is a standard technique for hiding such latencies. The computation instructions are interleaved with

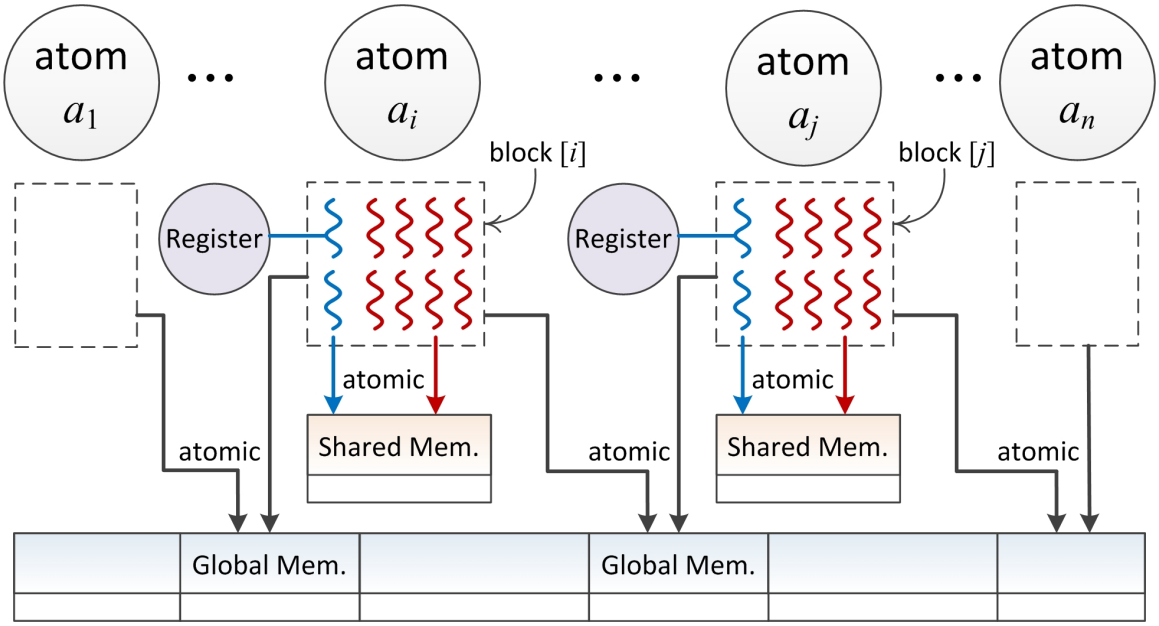


FIGURE 4.4.2: Thread execution model on the GPU.

memory access instructions, hence every time one thread is accessing the global memory the processor can switch the context to a different thread. Other optimization attempts include using local memories instead of global memories whenever possible, and avoiding multiple computations of constant parameters or variables that are used repeatedly.

4.4.3 GPU-Parallel Implementation

The second parallel implementation targets a fine-grained machine with a hierarchical memory architecture, i.e., one with a many-core many-thread GPU. Given n atoms, a linear grid of n blocks is generated, assigning one block per atom a_i $(1 \leq i \leq n)$. Each block is further divided into $N = |Q|$ threads, assigning one thread per sample point $\mathbf{q}_k \in Q$ $(1 \leq q \leq n)$ on the unit offset sphere. Prior to GPU kernel execution, the neighborhood information (i.e., the list of indices of all $a_j \in \mathbb{A}_i$ defined in (4.3.1) for

each atom $a_i \in \mathbb{A}$) is transferred from the CPU (i.e., *host*) memory to the GPU (i.e., *device*) memory. For each thread, the iteration over different neighbors is performed sequentially, similar to the CPU-parallel code presented in 4.4.2. The solvation energy and force components, a counter that keeps track of the number of overlapped sample points, and exposure states are initialized in the *shared* memory of the blocks, which require atomic operations for access safety by multiple threads, while sample point coordinates are stored in the *registers* that are local to each thread. Each sample point is tested sequentially against all neighbors to obtain $C_{i,k}$ $(1 \leq i \leq n, 1 \leq k \leq |Q_i|)$.

Once the exposure states are obtained for the original configuration of the neighbors, the thread loops over all neighbors one more time to examine the effect of their displacement along the 3 coordinate axes one at a time. Similar to the CPU-based implementation, whenever the pair of force components $\pm\delta F^{\text{cav}} = \pm 4\pi\gamma_i (R_i^{\text{off}})^2/(N\delta r)$ need to be added to the total solvation forces of two neighbor atoms a_i and a_j along the proper coordinate axis, the write operation that modifies $\mathbf{F}_i^{\text{cav}}$ of a_i happens atomically in the shared memory. This ensures mutual exclusion between threads of the same block. On the other hand, the write operation that modifies $\mathbf{F}_j^{\text{cav}}$ of a_j happens atomically in the global memory to ensure mutual exclusion between blocks of the same grid. Figure 4.4.2 shows the multi-threading scheme for the GPU-parallel algorithm. The algorithm is implemented using NVIDIA's compute-unified device architecture (CUDA). Kernel invocation is carried out synchronously within the default CUDA stream, hence synchronization between blocks is automatically guaranteed, while barrier synchronization is needed between threads of the same block.

GPU Optimization. The optimization attempts can be categorized as memory, execution, instruction, and flow-control optimization.

- Memory optimization is the most effective of all, as demonstrated by the results in the Section 4.5.2. In contrast to the CPU-parallel algorithm that makes most references through the cached global memory, the GPU-parallel algorithm transfers the coordinates, radii, solvation parameters, and neighbor index lists for each atom into the shared memory to minimize the number of global memory references. The variables that are exclusive to the threads, on the other hand, such as the exposure states or sample coordinates are allocated in the registers. However, the limited amount of shared memory and register resources are limited on the streaming multiprocessor (SM) imposes a restriction on the number of resident blocks on the SM and can adversely affect thread occupancy at any time during the simulation. Therefore, one needs to avoid excessive variable definitions within the scope of the GPU kernels.

- For execution level optimization, the kernels should be executed with proper granularity to maximize SM thread occupancy. Specifying a larger number of threads per block generally contributes to latency hiding, but is limited by the architecture as well as the on-chip memory resources. The number of threads is the same as the sample size $N = |Q|$ a proper choice of which is a trade-off between accuracy and performance.

- For instruction level optimization, the transcendental math functions are converted to their intrinsic alternatives that are executed on the special function units (SFU) of the CUDA cores.

- Flow-control optimization is realized by avoiding multiple execution paths within the same block, which might lead to thread divergence and serialization within the same warp. In particular, when checking for overlaps between neighbor atoms and sample points, the conditional (e.g., if/else/then) statements are set in such a way that one of the two execution paths is always null.

The near-optimal conditions are reached by successive experimentation and modification of the code. For more information regarding the GPU architecture and terminology, see Appendix C.2.

4.5 Results & Discussion

This section presents a preliminary assessment of the model and implementation enhancements from Protofold I [83–85] to Protofold II. The folding process is simulated and assessed at multiple levels, ranging from the formation of secondary structural elements (e.g., $\alpha-$helix coiling or $\beta-$strands formation) from an open chain to tertiary interactions between secondary elements or across larger domains that can be assumed to be rigid in real protein examples.

In Section 4.5.1 we discuss the impact of introducing solvation effects on the folding process of secondary structural elements starting from different initial conditions. We present some performance measures in Section 4.5.2 to validate the practical benefits of algorithmic improvements (e.g., coordinate hashing) as well as implementation improvements (i.e., CPU- and GPU-parallel computing). Finally, we look at a few real protein molecules in Section 4.5.3 and examine the energy variations in the neighborhood of the native structures.

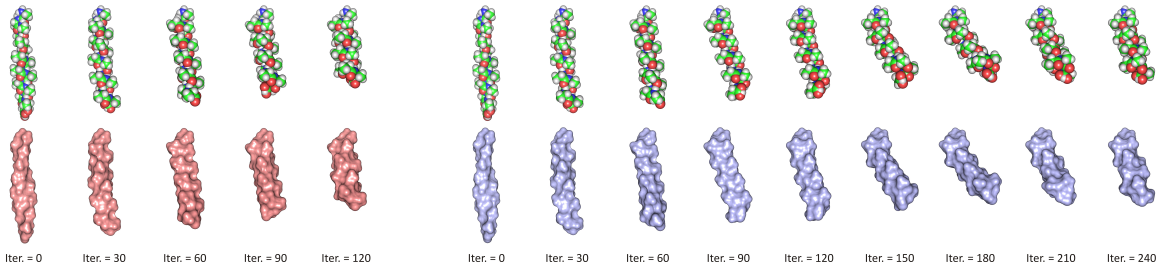


FIGURE 4.5.1: Left-handed $\alpha-$helix formation for a 15-residue polyalanine chain in vacuum (left) and in water (right) starting from $\phi_i^0 = \psi_i^0 = +10^\circ$ using Protofold II. Initial conditions and solvation effects dramatically affect the folding pathway.

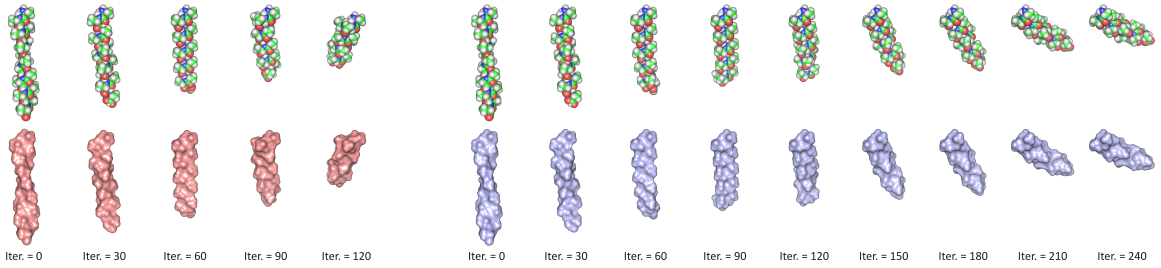


FIGURE 4.5.2: Right-handed $\alpha-$helix formation for a 15-residue polyalanine chain in vacuum (left) and in water (right) starting from $\phi_i^0 = \psi_i^0 = -10^\circ$ using Protofold II. Initial conditions and solvation effects dramatically affect the folding pathway.

4.5.1 The Folding Process

The simplest structural elements that are ubiquitous across many protein domains are $\alpha-$helices and $\beta-$strands. Here we start by considering simple test runs on relatively short (e.g., 10-20 residues long) peptide chains made of Ala residues (typically used as the benchmark AA type in its most common L-stereoisomeric form) to visualize their compliance into secondary structural elements.

Alpha Helix Formation. First, we run four tests on a 15-residue chain starting from two different initial conditions, for both of which we simulate the folding process without and with solvation effects taken into account:

1. Starting from the (slightly pre-coiled) initial conditions $\phi_i^0 = \psi_i^0 = +10^\circ$ for all $1 \leq i \leq 15$ the chain folds into a left-handed $\alpha-$helix as depicted in Fig. 4.5.1. The energy variation during KCM iterations is given in Fig. 4.5.3.

2. Starting from the (slightly pre-coiled) initial conditions $\phi_i^0 = \psi_i^0 = -10^\circ$ for all $1 \leq i \leq 15$ the chain folds into a right-handed $\alpha-$helix as depicted in Fig. 4.5.2. The energy variation during KCM iterations is given in Fig. 4.5.4.

The folding process in vacuum, i.e., without considering the solvation effects, emulates the behavior in the absence of the polar solvent, e.g., in membrane proteins extended along the nonpolar lipid bilayer or in secondary structural elements wrapped inside the hydrophobic core of globular proteins [91]. On the other hand, the folding process in water, i.e., with the presence of the solvation effects in addition to the intramolecular interactions, emulates the formation of elements that reside at the hydrophilic surface of globular proteins [91].

In the case of $\alpha-$helix formations in Figs. 4.5.1 and 4.5.2, the hand of the initial coil determines the hand of the final helix.[11] This is due to the gradient descent nature of the KCM search algorithm that tends to converge to different local minima depending on the initial state. The effects of the solvation are hard to observe in these examples with the energetically unchallenged helical structures due to proper stacking of the atoms favored by all considered effects. In both cases the van der Waals and solvation effects work in the same direction until the steric clash prevents the helix to coil further.

Figures 4.5.3 and 4.5.4 are plots of the free energy variations versus KCM iteration number for the four runs described above. Note that in all four cases (top and bottom plots) the solvation energy is evaluated and plotted, but only in two of them (bottom plots) its effects are applied to deform the chain. For both left and right-handed helix formation, the inclusion of solvation effects clearly changes the folding pathway and increases the number of iterations before convergence from around 150 to 300. However, in either case the solvation free energy changes are not as significant as those of intramolecular (particularly van der Waals) effects. Another important observation is that the right-handed $\alpha-$helix exhibits a notably more stable conformation than the left-handed $\alpha-$helix with about $\sim 40-50$ kcal per mol lower total free energy state—to be accurate, 43.8 and 45.9 kcal per mol for the entire chain, i.e., 2.9 and 3.1 kcal per mol per AA residue, without and with solvation effects, respectively. Although this is qualitatively consistent with the expectation of right-handed coiling being favored by L-alanine chains, the energy differences are higher than the ones reported in earlier studies (e.g., MD results in [99]). However, a meaningful comparison would require

[11] The surface visualizations can be deceiving where the right-handed helix appears to have a left-handed twist and vice versa. This is due to the transversal ridges and grooves formed in between the side chains [91].

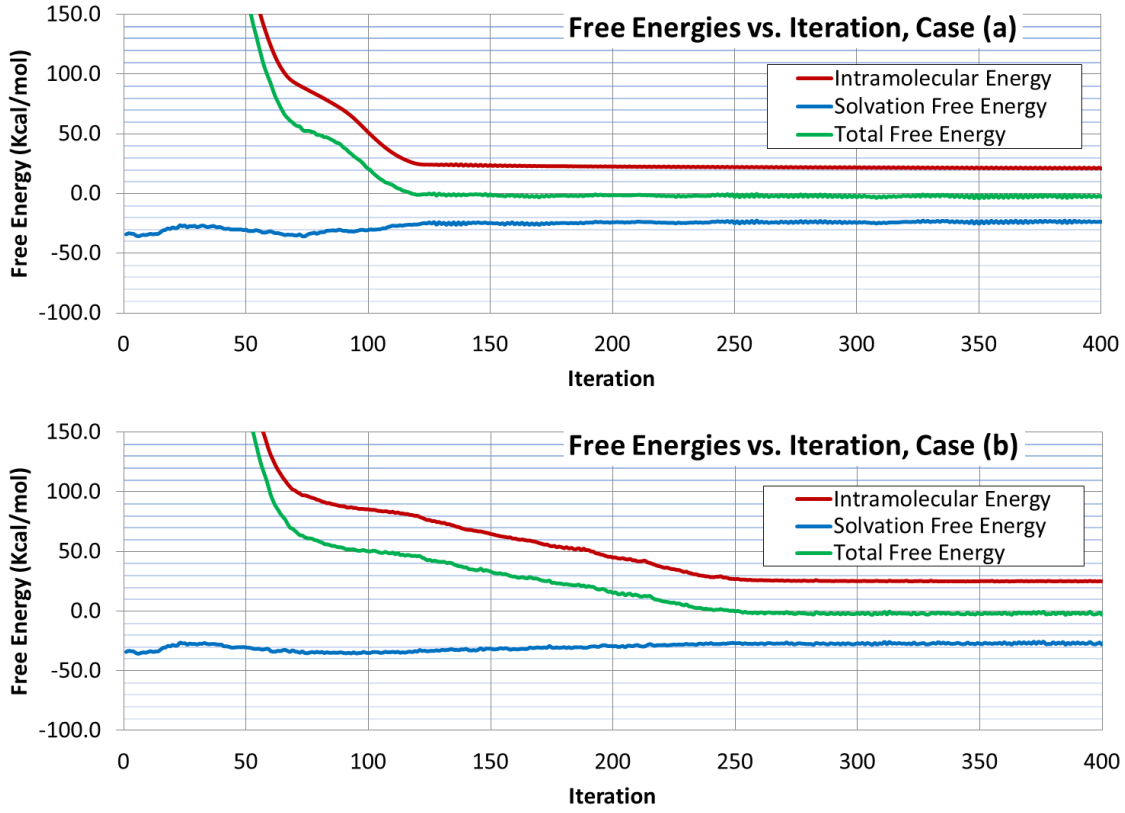


FIGURE 4.5.3: Free energy variations during folding of a 15-residue polyalanine chain into a left-handed $\alpha-$helix in vacuum (top) and in water (bottom) using Protofold II.

using identical simulation parameters, which is beyond the scope of this work.

The final dihedral angles for all 15 Ala residues corresponding to the folded (i.e., stable) conformations, obtained after a large enough number of iterations, are given in Table 4.5.1.

Ramachandran Plots. To examine the local effects of energetics, Ramachandran plots (for a pair of Ala residues in tandem) are generated by Protofold II using the energy-field presented in Section 4.2.2. The plots in Fig. 4.5.5 show the energy variations across different pairs (ϕ, ψ) of dihedral angles, without and with consider-

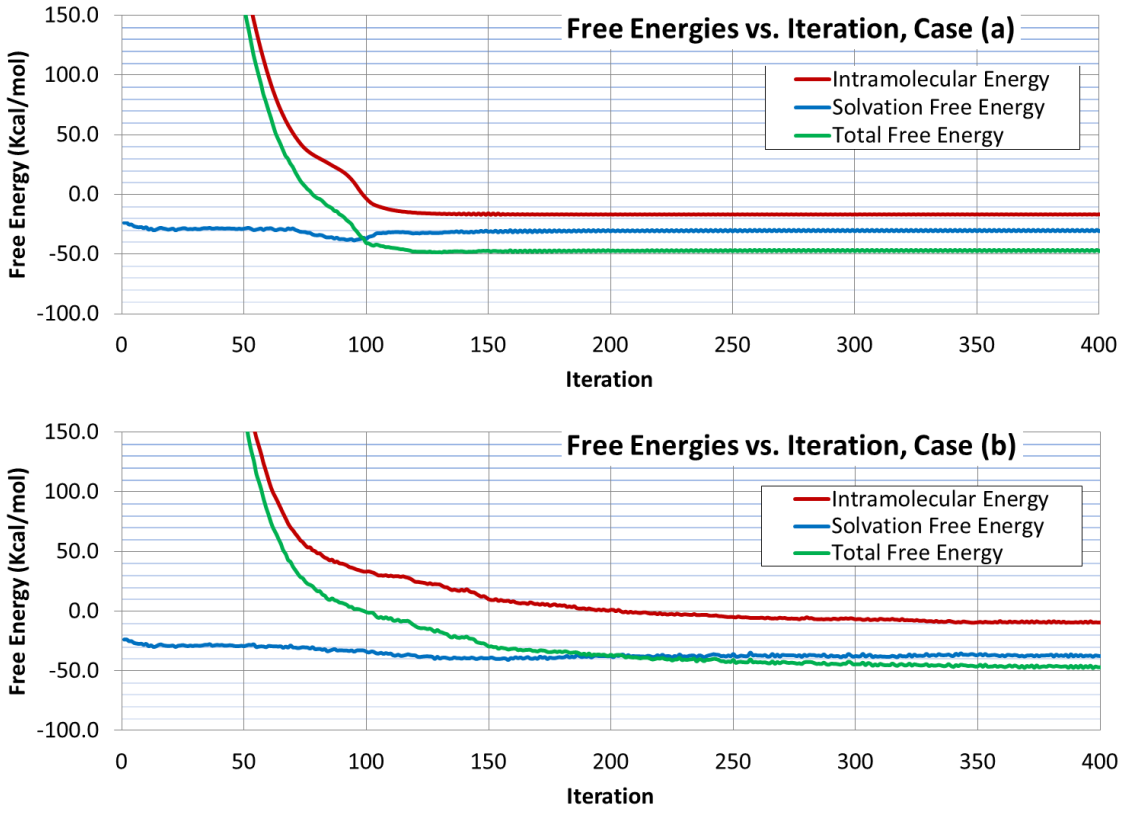


FIGURE 4.5.4: Free energy variations during folding of a 15-residue polyalanine chain into a right-handed $\alpha-$helix in vacuum (top) and in water (bottom) using Protofold II.

TABLE 4.5.1: Final conformations of a 15-residue polyalanine chain folded in vacuum using Protofold II.

	Left-handed coil $\phi_i^0 = \psi_i^0 = +10°$				Right-handed coil $\phi_i^0 = \psi_i^0 = -10°$			
	In vacuum		In water		In vacuum		In water	
i	$\phi_i(°)$	$\psi_i(°)$	$\phi_i(°)$	$\psi_i(°)$	$\phi_i(°)$	$\psi_i(°)$	$\phi_i(°)$	$\psi_i(°)$
1	+10.0	+118	−7.20	+21.5	−10.0	−59.1	+15.1	+54.4
2	+60.2	+53.9	+59.9	+47.6	−82.4	−41.1	−107	−47.1
3	+58.9	+36.8	+58.6	+40.0	−76.9	−24.8	−91.9	+11.3
4	+56.8	+45.4	+57.9	+45.1	−75.7	−34.4	−81.1	−34.4
5	+57.9	+42.2	+58.8	+40.3	−75.8	−28.4	−83.1	−23.8
6	+56.7	+43.7	+57.8	+43.6	−75.3	−31.3	−76.7	−32.1
7	+57.3	+42.7	+57.9	+43.6	−76.1	−29.3	−79.4	−28.9
8	+56.8	+44.0	+58.1	+41.6	−75.6	−30.5	−77.1	−28.9
9	+56.8	+42.3	+58.3	+42.1	−76.3	−29.2	−79.8	−29.2
10	+56.5	+45.5	+58.4	+43.3	−76.5	−29.7	−78.4	−28.3
11	+56.2	+42.4	+57.9	+41.7	−77.5	−29.5	−79.4	−28.6
12	+55.1	+48.4	+56.3	+46.0	−75.6	−33.5	−78.9	−28.7
13	+53.5	+45.5	+55.6	+45.0	−72.1	−39.0	−75.0	−33.4
14	+49.2	+59.4	+52.3	+52.6	−63.1	−44.4	−72.6	−40.0
15	+53.2	+69.8	+53.2	+72.7	−64.3	−53.5	−65.4	−48.0
Ave.	+53.0	+52.0	+52.9	+44.4	−70.2	−35.9	−74.1	−24.4

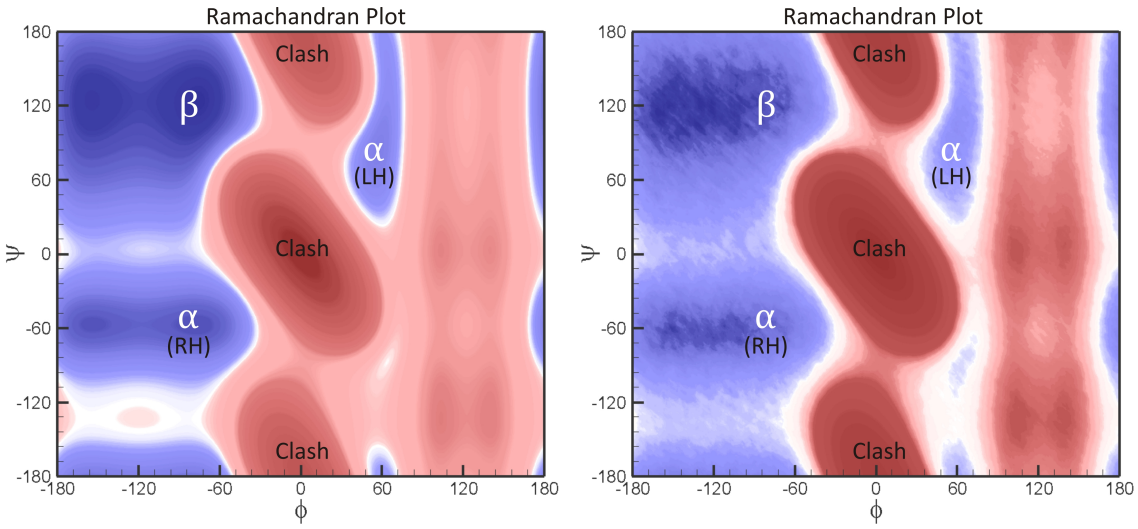


FIGURE 4.5.5: The Ramachandran plots for the energy variations of a pair of Ala residues in vacuum (i.e., without solvation effects) (left) and in water (i.e., with solvation effects) (right) using Protofold II.

ing the solvation effects. The vertical high-energy regions in the middle corresponds to prohibitive steric clashes between the atoms. The low-energy regions around it, on the other hand, correspond to the geometric relations between consecutive peptide planes that, when repeated for a segment of multiple residues along the chain, create secondary structural elements such as coiled $\alpha-$helices and flat $\beta-$strands. Although the solvation effects do not significantly alter the shape of the energy profile, there is a certain amount of noise added due to the discrete nature of the enumeration algorithm presented in Section 4.3.4.

To observe the effects of solvation, one needs to carry out more extensive simulations on larger data sets with different chain lengths and various initial conditions. We carried out KCM runs on 2,000 independent polyalanine chains of random lengths in the range $10 \leq m \leq 20$ starting from random initial angles in the range $-90^\circ \leq \phi_i^0, \psi_i^0 \leq +90^\circ$. The tests were run separately without and with considering

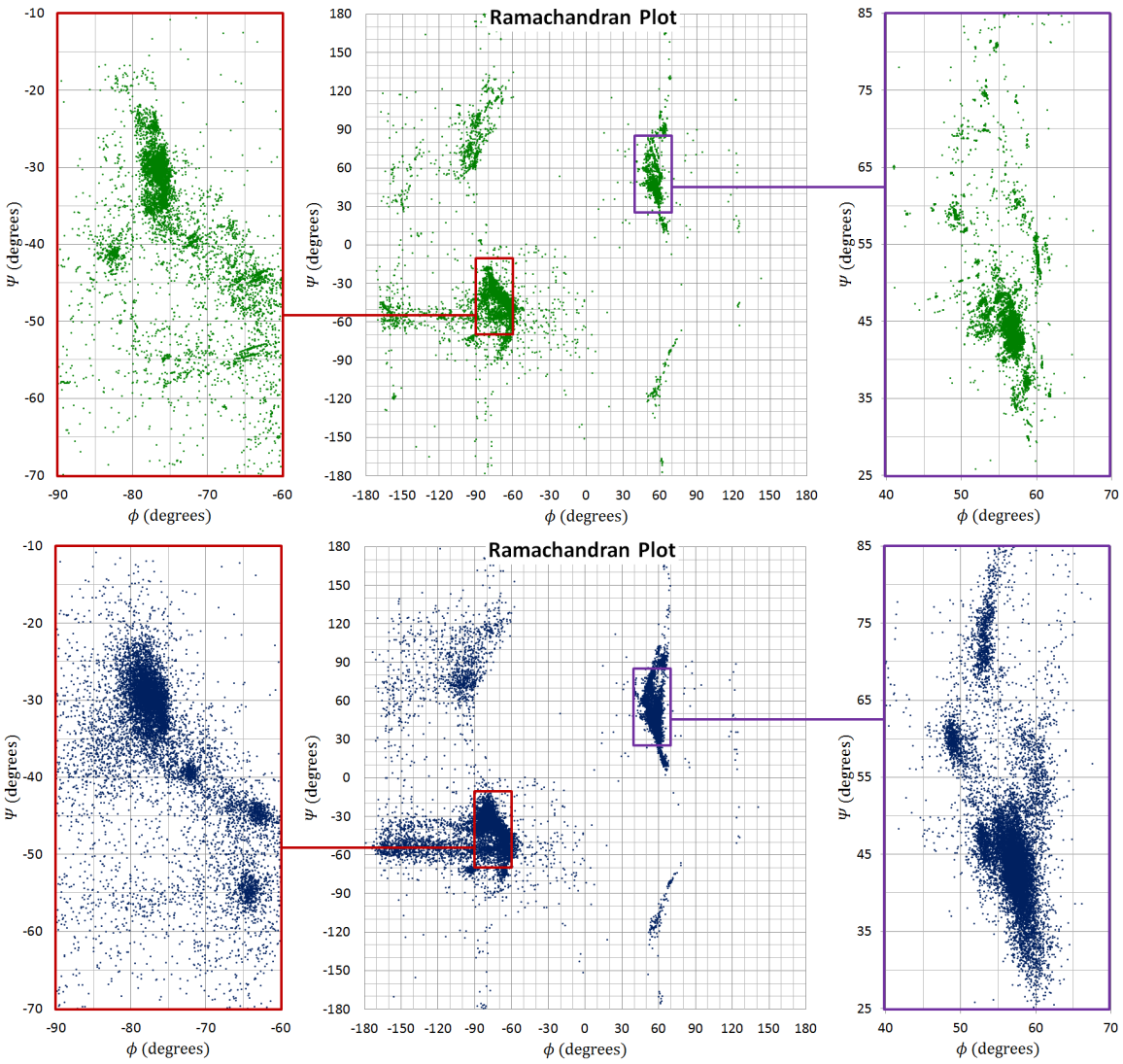


FIGURE 4.5.6: The Ramachandran plots for the folded backbone conformation of 2,000 10- to 20-residue polyalanine chains in vacuum (i.e., without solvation effects) (top) and in water (i.e., with solvation effects) (bottom) starting from random initial conditions $-90^\circ \leq \phi_i^0, \psi_i^0 \leq +90^\circ$ using Protofold II.

the solvation effects using the same random seed. The resulting dihedral angles for all 27,717 Ala residues of all chains[12] are plotted in Fig. 4.5.6. One can observe multiple concentration areas that clearly correspond to left- and right-handed $\alpha-$helices and flat $\beta-$strands, the former two helical folds being more populated on the plots. Zooming further on the two $\alpha-$regions reveals multiple local minima where the points are concentrated more, which correspond to different subtypes of $\alpha-$helices. Comparing the two plots in Fig. 4.5.6 reveals that the solvation effects do not significantly change the locations of the local minima. However, the energy profile is relaxed around the local minima where it has sharp cracks traced by the concentrated points along the valleys of the intramolecular energy landscape.

4.5.2 Computation Times

Next, we demonstrate the substantial performance improvements in Protofold II as a result of introducing the algorithms and data structures presented in Section 4.3. The running times are reported and compared on two computer systems, namely:

- **C-1**: Dell Precision T7500 workstation with an Intel® Xeon® E5645 CPU (12 cores, clock rate 2.40 GHz, and host memory 24 GB). The system is equipped with one NVIDIA Quadro® 4000 GPU (256 CUDA cores with compute capability (CC) 2.0 and device memory 2 GB).

- **C-2**: Dell Precision T7600 workstation with an Intel® Xeon® E5-2687W CPU (32 cores, clock rate 3.10 GHz, and host memory 64 GB). The system is equipped with two graphics cards: a NVIDIA Quadro® K5000 GPU (1,536

[12]The angles for the first AA residue of all chains are eliminated as outliers since they differ dramatically from those of the subsequent residues due to the anchoring at the N-terminus (see Table 4.5.1).

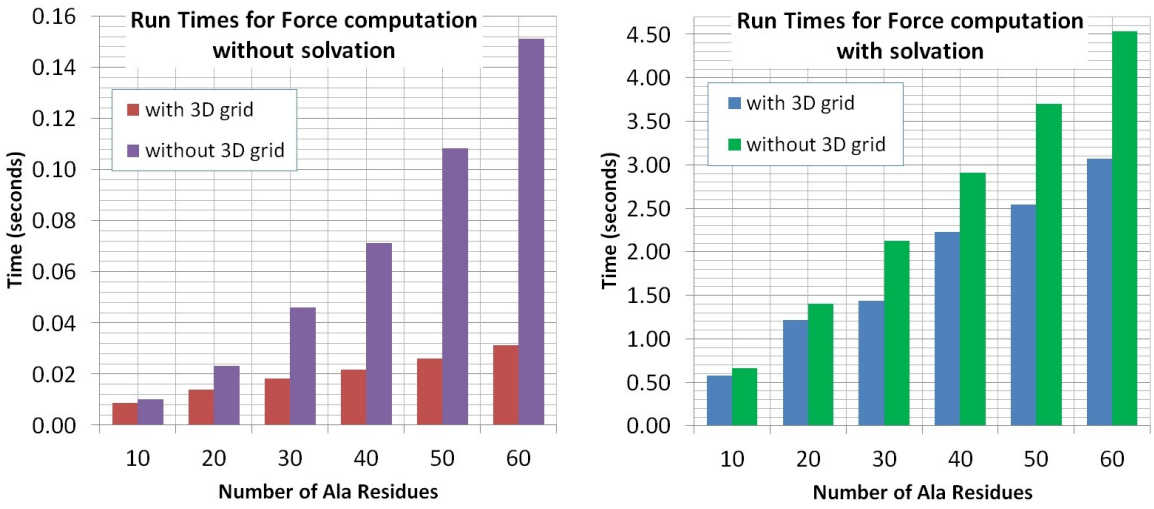
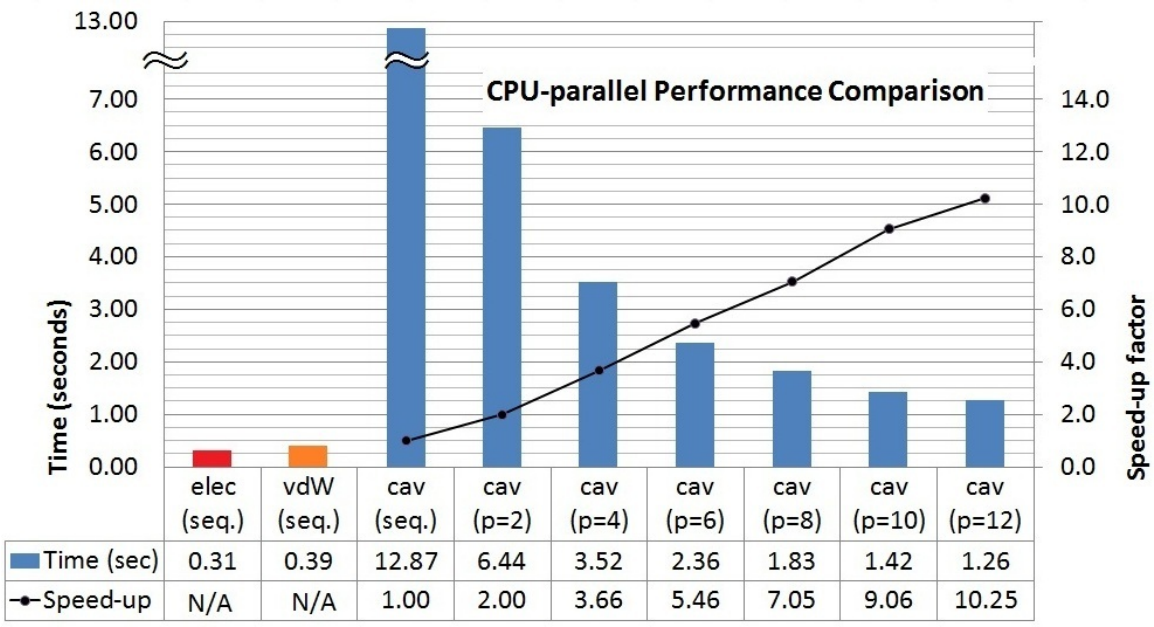


FIGURE 4.5.7: The effect of 3D hashing on the force computation times of a 60-residue polyalanine chain with and without solvation effects (on **C-1**).

CUDA cores with CC 3.0 and device memory 4 GB) and a NVIDIA Tesla® K20C GPU (2,496 CUDA cores with CC 3.5 and device memory 5 GB).

Effect of Hashing. Figure 4.5.7 shows the running times of the force computation step (on **C-1**) in a single KCM iteration for folding polyalanine chains of different lengths with and without 3D hashing presented in Seciton 4.3.2 both in vacuum (i.e., without considering solvation effects) and in water (i.e., with considering solvation effects). In both cases, the results show a significant reduction in the running times with hashing (e.g., up to 4.6× in vacuum and 1.5× in water for $m = 60$ Ala residues), and the difference scales with the size of the molecule. Nevertheless, the solvation force computations remain the bottleneck of the simulation (using sequential CPU implementation) and adversely affect the speed-up gained from hashing.



	elec (seq.)	vdW (seq.)	cav (seq.)	cav (p=2)	cav (p=4)	cav (p=6)	cav (p=8)	cav (p=10)	cav (p=12)
Time (sec)	0.31	0.39	12.87	6.44	3.52	2.36	1.83	1.42	1.26
Speed-up	N/A	N/A	1.00	2.00	3.66	5.46	7.05	9.06	10.25

FIGURE 4.5.8: Sequential running times for electrostatic, van der Waals, and solvation forces, and CPU-parallel running times of the latter for a 1,200-residue polypeptide chain (on **C-1**).

Parallel Computing. The first 3 columns on the left in Fig. 4.5.8 show the sequential CPU running times (on **C-1**) for the electrostatic, van der Waals, and nonpolar solvation forces in a single KCM iteration. The results were obtained for a polypeptide chain composed of $1,200$ residues that are randomly selected from Ala, Cys, and Ser AAs—the latter two being equivalent to replacing the H in the nonpolar Ala side chain with SH and OH, respectively, resulting in polar side chains. It is clearly observed that with a single CPU core, the first two terms take a very small fraction of the total time (around $5-6\%$ of the total force computation time per iteration) and the solvent effects are clearly the bottleneck.

The same column chart in Fig. 4.5.8 also shows the running times and corresponding speed-ups (on **C-1**) for the CPU-parallel computation of the solvation force using up to 12 CPU cores. An almost linear speed-up is achieved by increasing the number

of processors (e.g., $\sim 10\times$ with 12 cores). However, the solvation force calculation is still about an order of magnitude slower than that of the other two force types.

Figure 4.5.9 compares the running times and corresponding speed-ups (on **C-1**) of the CPU- and GPU-parallel implementations for polypeptides of different lengths, ranging from $m = 200$ (i.e., $\sim$ 2K atoms) to $1,200$ AAs (i.e., $\sim$ 13K atoms). To depict the importance of memory optimization presented in Section 4.4.3, the running times are shown for two different cases; namely, one that uses global memory for communications between all threads of all blocks, and the optimized code making extensive use of shared memories for communications between threads within the same block. It is interesting to note that when the GPU shared memory is not utilized, the results do not show an improvement over the 12-core CPU implementation due to large global memory access latencies. However, proper shared memory usage results in a huge performance improvement that scales by protein size, e.g., from $20\times$ for $m = 200$ AAs to $70\times$ for $m = 1,200$ AAs with respect to the sequential run on a single CPU.

The computations are repeated for even larger molecules in Fig. 4.5.10, ranging from $m = 1,000$ AAs (i.e., $\sim$ 10K atoms) to $6,000$ AAs (i.e., $\sim$ 64K atoms), enabled by leveraging a more powerful machine (i.e., **C-2**). For the case of $m = 1,000$ AAs, **C-2** yields a two-fold speed-up on the CPU and a three-fold speed-up on the GPU compared to **C-1**. As depicted in Fig. 4.5.10, significantly higher and more consistent CPU speed-ups of 16-18$\times$ and GPU speed-ups of 90-100$\times$ (for Quadro® K5000) and 270-290$\times$ (for Tesla® K20C) are observed. These observations imply proper scalability of the data-parallel implementation with molecular size.

Even for molecules with tens of thousands of atoms, each force-field computation takes only a fraction of a second per iteration. This enables fast KCM simulation

of folding for large proteins over extended periods of time via Protofold II, which wouldn't be tractable via Protofold I [83–85].

4.5.3 Real Examples

Having considered the folding of secondary structural elements with complete flexibility (i.e., each peptide plane being treated as a separate rigid body), we proceed to study the tertiary interaction between larger rigid units in real proteins. We report on the following case studies:

- The interactions between multiple rigid $\alpha-$helices that rotate around flexible loops within the containing motifs/domains are considered. Two examples from the PDB are used for this purpose: Myoglobin (PDB: 1TES) and Troponin-C (PDB: 2JNF).

- The interactions between multiple rigid domains that are connected via flexible loops within the containing monomeric unit are examined. The example of Gamma-B Crystallin (PDB: 1GCS) is used for this purpose.

These PDB structures that are obtained from X-ray crystallography (e.g., 1TES and 1GCS) do not contain the H atoms, hence are first preprocessed using Duke University's MolProbity server [30,42] which predicts and adds the H atoms to the coordinates information before importing the structure into Protofold II. The PDB structures that are obtained from nuclear magnetic resonance (NMR) spectrometry (e.g., 2JNF), on the other hand, already contain the H atoms positions. The size of the molecules, i.e., the number of AA residues m and the number of atoms n' and n with and without the H atoms included, respectively, are given in Table 4.5.2.

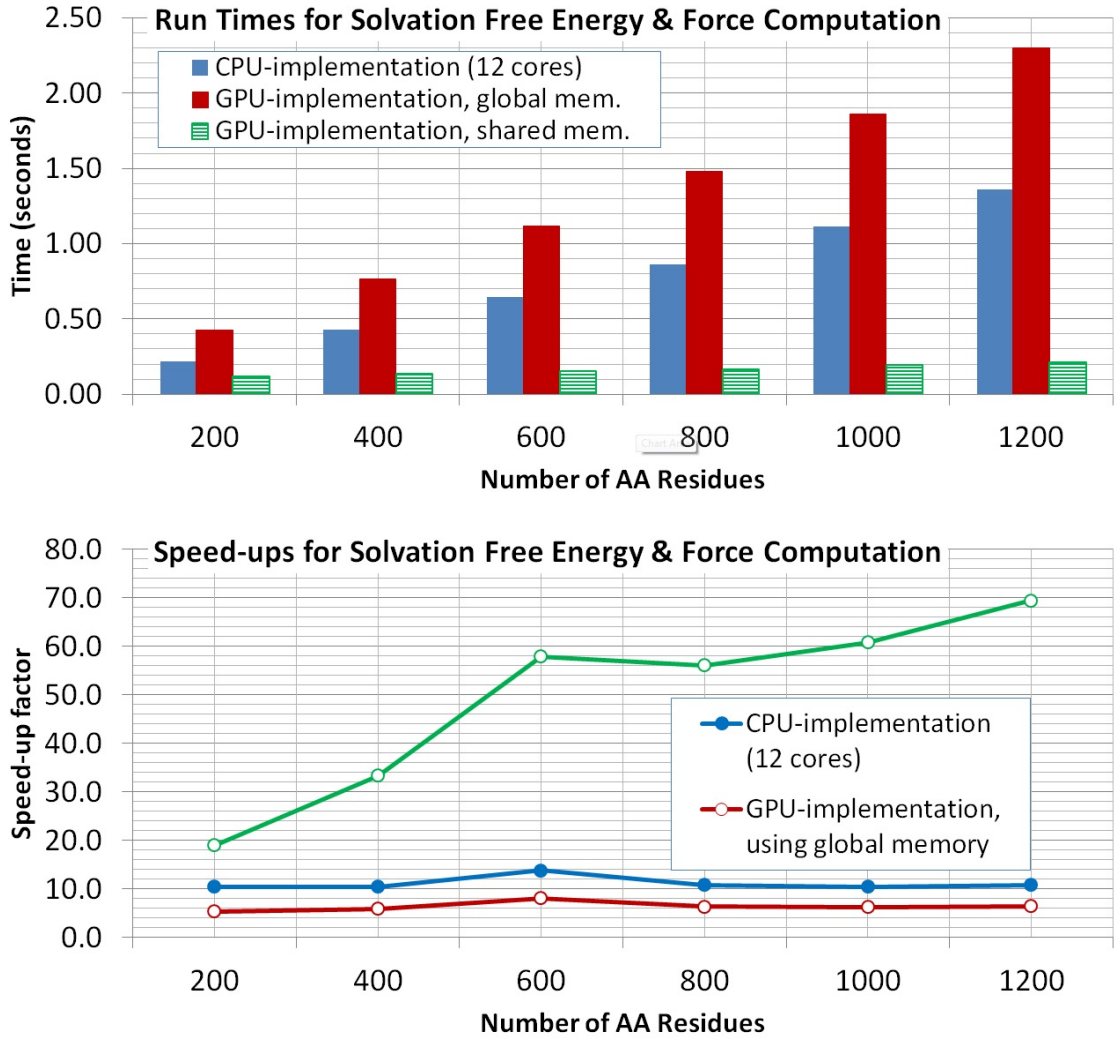


FIGURE 4.5.9: CPU- and GPU-parallel running times (top) and speed-ups (bottom) with and without memory optimization for polypeptide chains of various lengths (on **C-1**).

TABLE 4.5.2: The size of the analyzed protein examples.

Protein Name	PDB Code	m	n'	n
Myoglobin	1TES	154	1,231	2,478
Troponin-C	2JNF	158	1,232	2,401
Gamma-B Crystallin	1GCS	174	1,474	2,844

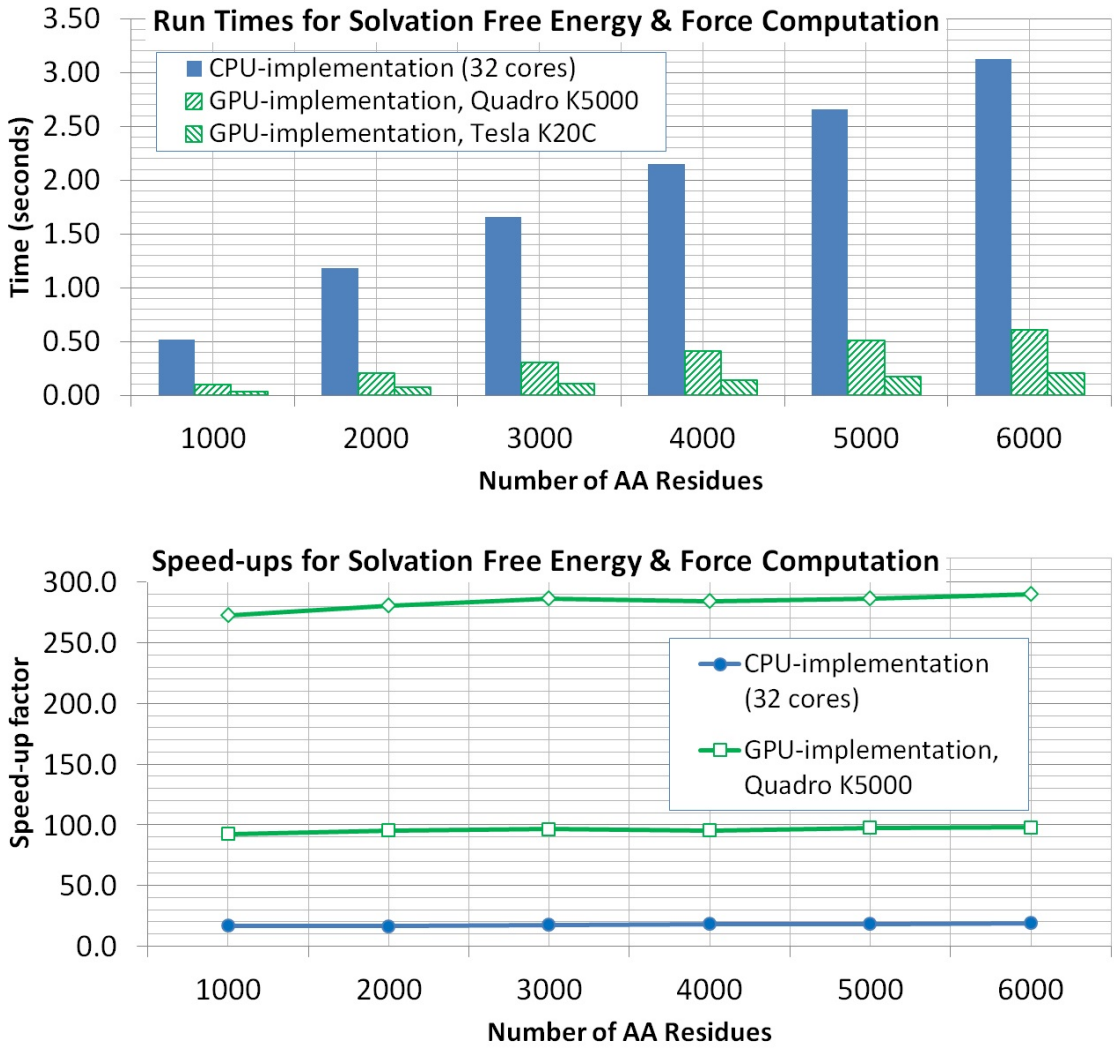


FIGURE 4.5.10: CPU- and GPU-parallel running times (top) and speed-ups (bottom) with memory optimization on two GPUs for polypeptide chains of various lengths (on **C-2**).

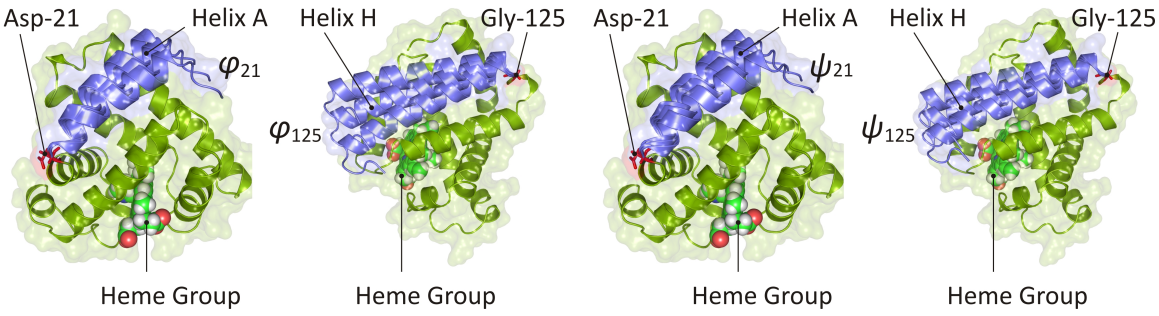


FIGURE 4.5.11: Myoglobin (PDB: 1TES) hinged at Asp-21 and Gly-125 for variations in ϕ_{21}, ϕ_{125} (left) and ψ_{21}, ψ_{125} (right).

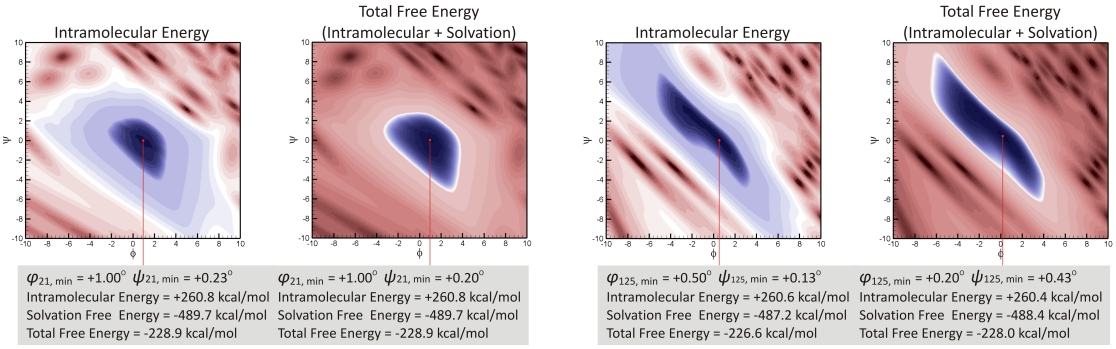


FIGURE 4.5.12: Intramolecular versus total (i.e., solvation included) free energy landscape for Myoglobin (PDB: 1TES) in the vicinity of the native conformation versus variations in (ϕ_{21}, ψ_{21}) (left) and (ϕ_{125}, ψ_{125}) (right).

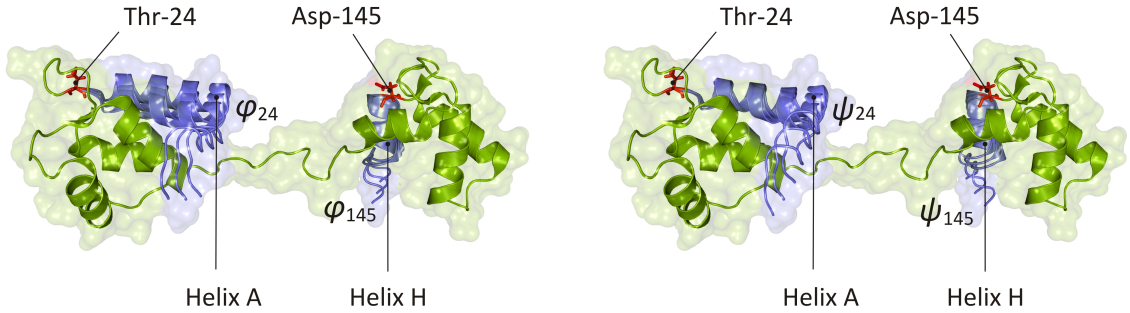


FIGURE 4.5.13: Troponin-C (PDB: 2JNF) hinged at Thr-24 and Asp-145 for variations in ϕ_{24}, ϕ_{145} (left) and ψ_{24}, ψ_{145} (right).

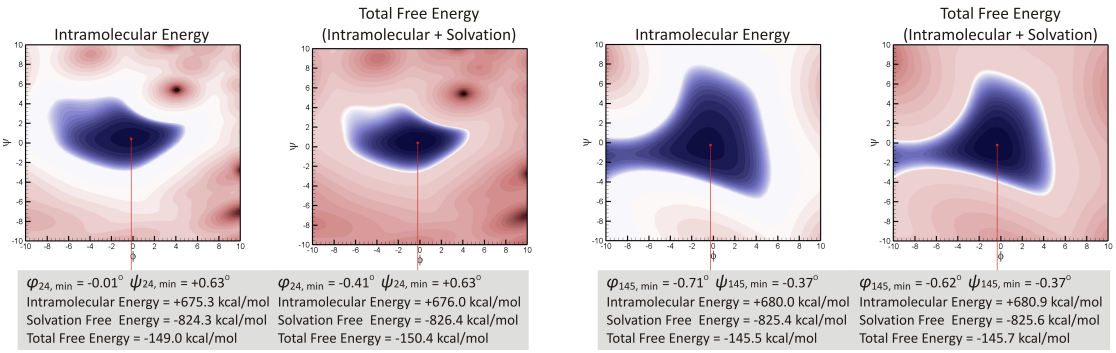


FIGURE 4.5.14: Intramolecular versus total (i.e., solvation included) free energy landscape for Troponin-C (PDB: 2JNF) in the vicinity of the native conformation versus variations in (ϕ_{24}, ψ_{24}) (left) and (ϕ_{145}, ψ_{145}) (right).

Secondary Structural Interactions. Let us first consider the energy variations when an $\alpha-$helix of an $\alpha-$domain is reoriented from its stable conformation with respect to the rest of the bundle, as illustrated in Figs. 4.5.11 and 4.5.13.

Myoglobin (PDB: 1TES) is an oxygen binding muscle protein that is composed of a single 'globin fold' domain, which is an $\alpha-$domain motif consisting of a bag of 8 $\alpha-$helices per domain (denoted A through H) arranged at $\sim +90°$ and $+50°$ angles with respect to each other, as shown in Fig. 4.5.11. This arrangement creates a hydrophobic pocket in the interior that wraps the stabilizer co-factor known as 'heme group' [125]. Assuming that the helices are rigid, we examine the energy variations due to dihedral rotations at the loops that connect the two end $\alpha-$helices; namely, local changes in (ϕ_i, ψ_i) for $i = 21$ and $i = 125$ where the A and H helices are hinged, respectively.

Troponin-C (PDB: 2JNF) is a calcium binding muscle protein that is composed of two 'EF-hand' domains, which are $\alpha-$domain motifs consisting of a bundle of 4 $\alpha-$helices per domain (denoted A through H) arranged in an up-and-down anti-parallel conformation [125], as shown in Fig. 4.5.13. There is a long pseudo-helical

segment that connects the two globular domains. Assuming that the helices are rigid, we examine the energy variations due to dihedral rotations at the loops that connect the two end $\alpha-$helices; namely, local changes in (ϕ_i, ψ_i) for $i = 24$ and $i = 145$ where the A and H helices are hinged, respectively.

Figures 4.5.12 and 4.5.14 show the free energy variations in vacuum (i.e., without considering solvation effects) and in water (i.e., with considering solvation effects) for the two protein domains as the end $\alpha-$helices A and H of each are rotated within a range of $\pm 10^\circ$ of the native (ϕ_i, ψ_i) at the hinged loops.[13] In all four cases the energy model of Protofold II exhibits a local minima within $\pm 1^\circ$ of the native conformation. We also observe that the solvation effects contribute a significant amount to the total free energy; however, the SASA variations are so small in the considered neighborhood that the location of the local minima is almost unchanged. The van der Waals effects appear to be dominant in this neighborhood, manifested as the shape complementarity between the ridges and grooves of the mobile $\alpha-$helix and those of the other helices in the bundle [49]. However, when variations across larger angular ranges are considered, the solvation effects are expected to play a more determining role.

Tertiary Structural Interactions. Lastly, we consider the energy variations when a rigid domains of a protein is reoriented with respect to another domain against which it is packed into a stable structure.

Gamma-B Crystallin (PDB: 1GCS) is an eye-lens protein that is made of two similar domains that are 40% identical in sequence [125]. Each domain is composed

[13] The colormaps are generated using a nonlinear but consistent contouring scheme.

of two anti-parallel $\beta-$sheets each made of 4 $\beta-$strands with the same arrangement topology [125], as shown in Fig. 4.5.15. Assuming that these domains are rigid, we analyze the energy variations due to rotating one domain with respect to the other at one of the residues that belong to the connecting loop (e.g., $i = 81$). To observe the global effects of solvation, we allow both dihedral angles (ϕ_{81}, ψ_{81}) to vary over the entire range of $\pm 180°$ from the native values. To facilitate visualization, this time we consider only one angle's variation at a time.

Figures 4.5.16 and 4.5.17 show the variations of the different energy terms.[14] It appears that the van der Waals effects are dominant in determining the profile of the energy well near the native conformation, which can be attributed to the extensive contact interface between the two domains in Fig. 4.5.15. The electrostatic and solvation effects substantially change the energy landscape thus can dramatically affect the folding pathway. However, they do not cause a significant change in the location of the energy minimum. Once again, the solvation energy is noisier due to the discrete nature of the enumeration algorithm in Section 4.3.4. Another interesting observation is that the electrostatic energy has discontinuities due to the cut-off approximation when pairs of atoms are farther than 9.0 $\mathring{A}$. Although it does not seem to affect the minimum location in this example, larger cut-off distances might be necessary for analyzing large proteins, since the accumulation of the pairwise errors grows quadratically with the number of atoms.

[14]To enabel logarithmic plots, each energy ordinate is offset by a constant value to shift its minimum to (the arbitrary positive value of) 100 kcal per mol.

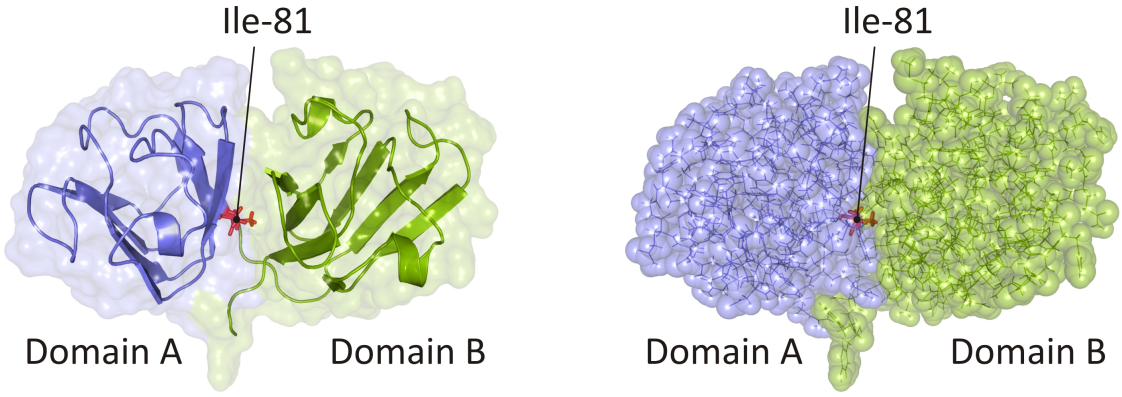


FIGURE 4.5.15: Gamma-B Crystallin (PDB: 1GCS) hinged at Ile-81 for variations in ϕ_{81} and ψ_{81}, one at a time.

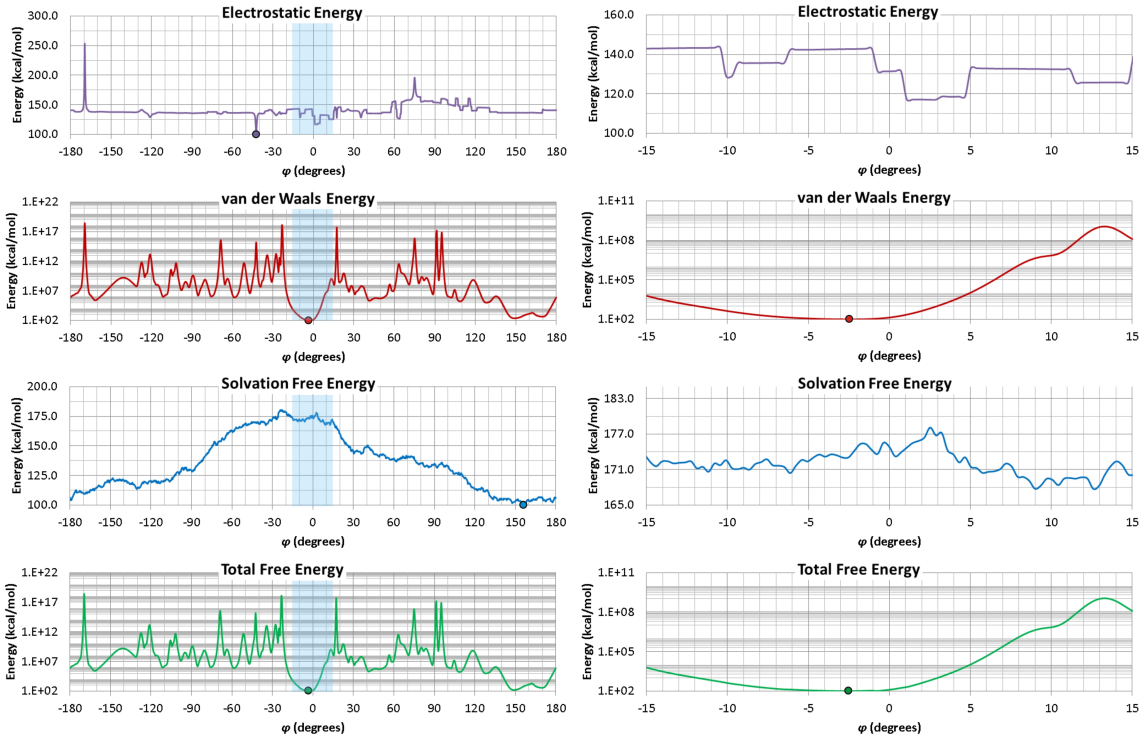


FIGURE 4.5.16: Energy variations for Gamma-B Crystallin (PDB: 1GCS) versus changes in ϕ_{81} for fixed native ψ_{81}.

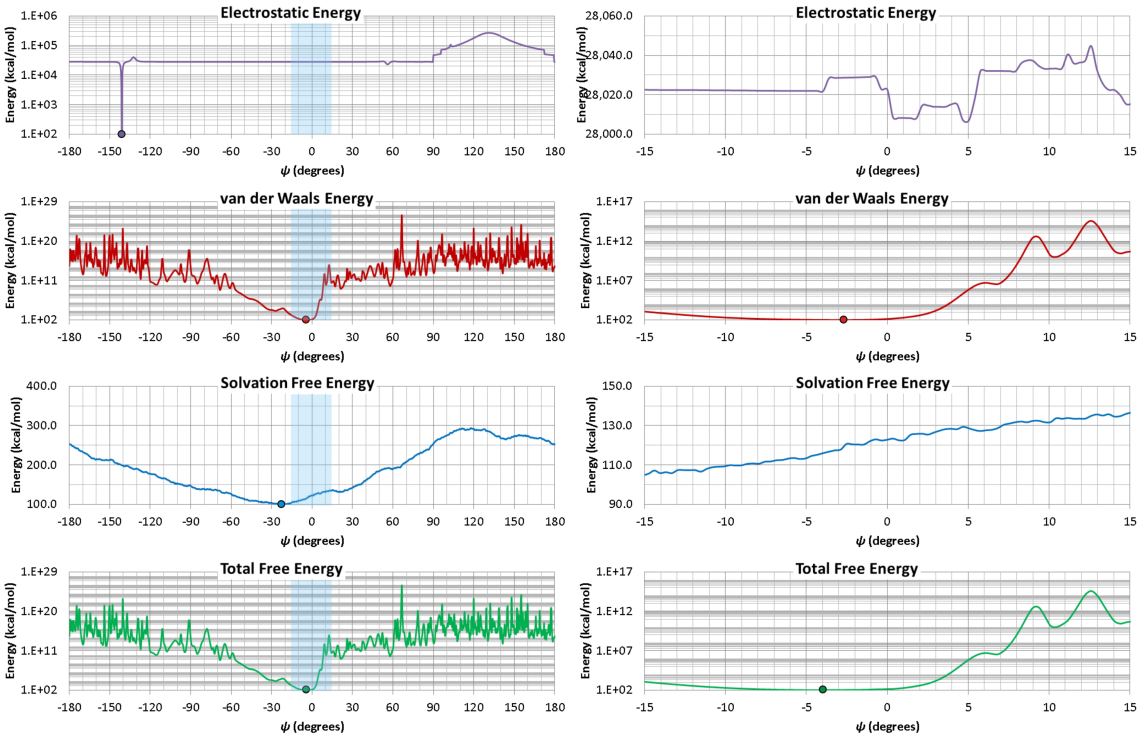


FIGURE 4.5.17: Energy variations for Gamma-B Crystallin (PDB: 1GCS) versus changes in ψ_{81} for fixed native ϕ_{81}.

Chapter 5

Approximating Net Interaction Among Rigid Bodies, Resulting From Pairwise Interactions Between Their Constituents

5.1 INTRODUCTION

At the very heart of every static or dynamic simulation, lies the evaluation of physical interactions such as different forms of energy and force. Predicting trajectories in a complex many-body system demands numerical solving of equations of motion, which in turn involves evaluation of pairwise interactions between individual objects at different time steps. Oftentimes, a large portion of the processing time is dedicated to pairwise interactions, which takes quadratic time, presuming that each particle is influenced by every other particle present in the system. However, the nature of every particular problem enforces a set of restricting conditions which perhaps can

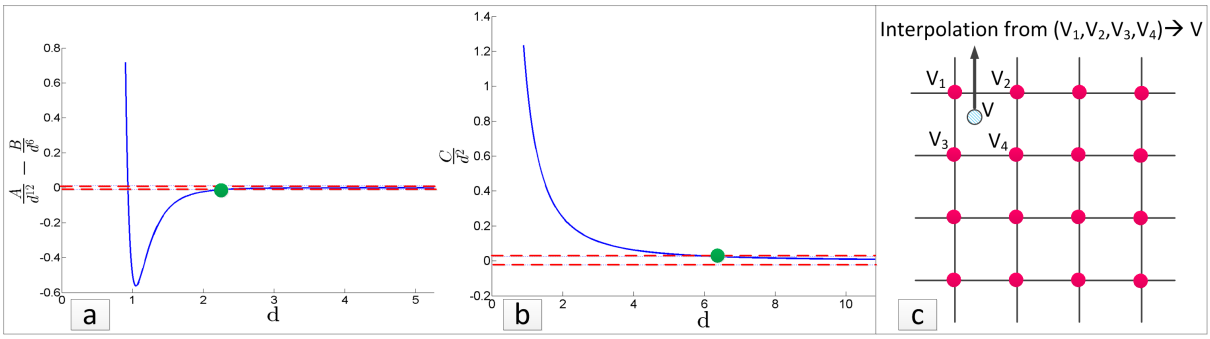


FIGURE 5.1.1: Left: trend of pairwise interaction vs. pairwise distance for (a) van der Waals energy (b) electrostatic energy and/or gravity force ($A = 1$,$B = 1.5$ and $C = 1$). Within the dotted lines, pairwise interaction can be neglected and the green dot specifies the cut-off distance. Right: Interpolating from mesh values.

be utilized to simplify the original problem. In the following, major categories of simplifying techniques are reviewed.

5.1.1 Ignoring Far Interactions

The profiles of several distance-dependent interactions suggest that the mutual effect of 'far' particles on each other can be neglected where, 'far' implies farther than a so-called cut-off e which varies depending on the type of interaction. Figures 5.1.1a and 5.1.1b show this for different types of physical interactions. Assuming that electrostatic and van Der Waals force interactions between pairs of atoms fade away as their distance exceeds 9 and 5 Angstroms, respectively, [160] captures neighborhood information in an atomic ensemble using a *3D hash table*, leading to an $n-$fold reduction in force computation time.

5.1.2 Using Mesh Values and Interpolation

In many physical applications, the interaction profile does not experience abrupt changes anywhere throughout the domain under study. Particle-in-cell methods lay out a mesh on the computational domain to interpolate (Fig. 5.1.1c) the values of pairwise interactions using the mesh values [67, 75]. Their computational complexity is $O(n + mlogn)$, where n and m are the numbers of particles and mesh points, respectively [75]. In order to address the limited resolution provided by the mesh, the P^3M method introduced in [75] computes the short-range interactions directly, while for far-field interactions uses the mesh values. A similar approach is taken in [123], by replacing the mesh with a tree. By stating the potential at each point as a sum of three components, namely, far, near and external, [67] uses multi-pole expansions to reduce the computation complexity down to $O(n)$.

5.1.3 Exploiting Rigidity

Oftentimes, it is observed in physical systems that groups of objects are lumped together as rigid bodies. Examples of this phenomenon can be seen in hierarchic protein folding instances where the interaction between almost rigid domains directly influences the folding pathway [11], interaction between macromolecules [77, 108, 181] for drug design purposes, self-assembly of nano-particles [186] for drug design and drug delivery applications, design of smart materials [60] and bio-sensors [102], as well as the interaction between stellar clusters [51]. In such cases, we look at the motion of each ensemble as a whole. Also, rather than evaluating single pairwise interactions, we are more interested in their net effect. The assumption of rigidity can be exploited in different ways:

Faster Evaluation of Kinematic Parameters

Rigid body constraint algorithm is incorporated in [118] into a GPU-accelerated MD to speed up the the numerical solving of equations of motion. Distance and angular holonomic constrains are enforced in [45] during the molecular simulations. Also, [47] suggests a robust parallelizable constraint method for molecular simulations. These methods however do not offer techniques for reducing the computational complexity associated with interaction computations.

Faster Net Interaction Evaluation by Avoiding Pairwise Evaluations within Rigid Bodies

In the brute force approach, computing the net interaction between a pair of rigid bodies is accomplished by simply adding up all the pairwise interactions. This can be however computationally improved when rigid domains are identified within the system. In an effort to computationally benefit from the rigidity condition, [87, 160] modeled protein molecule as a kinematic chain with groups of atoms lumped together, acting as rigid bodies or links, leading to speedups in computing forces and motion tracking in their simulation, by neglecting the internal forces. However, since the number of rigid bodies were proportional to the number of atoms, the computational complexity remained dependent of the number of atoms. The *Chain Tree* data structure introduced in [100] stores the distance status of pairs in a hierarchy, allowing fast inquiry and update for them, avoiding interaction evaluations within rigid sections and resulting in logarithmic reduction of time complexity (still dependent on the number of particles). Notice that, as long as the net effect of interactions on each individual particle is of concern, the minimum reachable computational complexity

is from the order of $O(n)$, n being the number of particles present in the system.

Faster Net Interaction Evaluation by Approximating Attributes of Rigid Body Pair

As long as the pairwise effect of interest is distance dependent, the net interaction can be expressed only as a function of the relative pose of the two bodies, plus a description of geometry at a reference relative pose. The number of independent variables needed for articulating the pose equals 6 in the most general case, with 3 translational and 3 rotational variables. However, this number can be as low as 1 if specific restraints are enforced such as the case studied in [186], where the two alpha helices are hinged together by means of a revolute joint. This suggests that unlike the general $n-$ body problem (where the net effect of interactions on each particle must be tracked), a computational complexity that is independent of n is in theory achievable, due to the fact that number of parameters needed to describe the state of the systems collapses from $O(n)$ to $O(1)$ (= the number of relative pose parameters). Although for certain special cases with restrained geometries, closed form simplifications of the interaction functions can be suggested, such as the case of gravity force between earth and a mass on its surface (where the earth is taken as a sphere and the mass on the surface is taken as a point), a straightforward formulation does not seem to exist for the general unrestrained geometry (i.e. between arbitrary shapes), as we may end up with expressions with quadratic number of terms, each of which referring to a single pairwise interaction. In [126], a method is proposed for evaluation of long range forces and moments between rigid bodies. For a prevalent form of the interaction function, where force is a negative integer power of distance, the pairwise interaction in [126] is approximated by obtaining

the binomial series expansion of the force function followed by neglecting the terms that become insignificant when the two objects get "far enough" from each other. The method offers a computational complexity of interaction evaluation which is independent of the number of particles present in the simulation. However, observe that the method is limited to specific forms of force function ($F \propto 1/d^n$, where d is distance and n is a positive integer number). Therefore, other forms of force, for instance the widely used linear and non-linear spring models [5, 92] for describing the interaction between particles do not fall into the scope of application of their method. In addition, the method only addresses the computation of long-range interactions and therefore fails to compute the net interaction when the two objects become closer than a threshold distance. Therefore, short-range interactions which play an important role in determining the overall attributes of the system must be treated separately with conventional approaches.

In this chapter, we attempt to address the shortcomings of the existing methods in exploiting the rigidity condition. We propose to approximate the pairwise interaction function using a linear predictor function, in which the basis functions have separated forms [17], i.e. the variables that describe local geometries of the two rigid bodies and the ones that reflect the relative pose between them are split in each basis function. Doing so facilitates certain summation operations on the pairwise interactions during a preprocessing step, yielding fast evaluation of instantaneous net interaction whenever required. The multivariate pairwise interaction function is approximated by one that has the following *separated* form [17]: $f \approx \sum_{k=1}^{r} \beta_k g_k h_k$, where h_k only concerns the relative pose of the two rigid bodies, g_k is only descriptive of geometry at a reference relative pose, r is the separation rank (Fig. 5.1.2) and β_k is the regression coefficient. The advantage of this is that, now, we can collect all the

terms with similar h_k and sum over all the corresponding $\beta_k g_k$ values (which can be obtained from the geometry at the reference relative pose) once, in a preprocessing step to attain a set of *characteristic parameters*. Then, computing the net interaction requires evaluation of only $O(r)$ terms, i.e. $\{h_k | 1 \leq k \leq r\}$, instead of one evaluation per each pair of particles. Using this method, the quadratic number of pairwise interaction evaluations ($O(rMN)$, M and N, being the numbers of particles of the two bodies), in computing the net interaction at each relative pose, is replaced with one-time quadratic operations in preprocessing ($O(MN)$) plus constant pose function evaluations at each pose ($O(r)$), where r is independent of M and N and is only a function of the accuracy and efficiency of the used approximation method.

The two steps for arriving at a linear regression that approximates a multivariate pairwise interaction[1] are: (1) finding appropriate basis functions (i.e., $\{h_k, g_k | 1 \leq k \leq r\}$) (2) adjusting the regression coefficients (i.e., $\{\beta_k | 1 \leq k \leq r\}$). Note that the accuracy of pairwise interaction has direct influence on the accuracy of the resulting net interaction.

Several different approaches can be found in the literature for approximating multivariate functions. A review of finite sums decomposition methods in mathematical analysis can be found in [130]. In [26], decomposable functions of several variables are studied. The paper deals, in particular, with a function of three variables and claims that many of the results are extendable to more than three variables. The algorithm presented in [17] estimates a function of many variables from scattered data by approximating it as a sum of separable functions. The method is linear in the number of data points as well as the number of variables, which makes it suitable for large data sets in high dimensions. Also, [117] gives a method for decomposing smooth function

[1] for functions that admit this variable decomposition

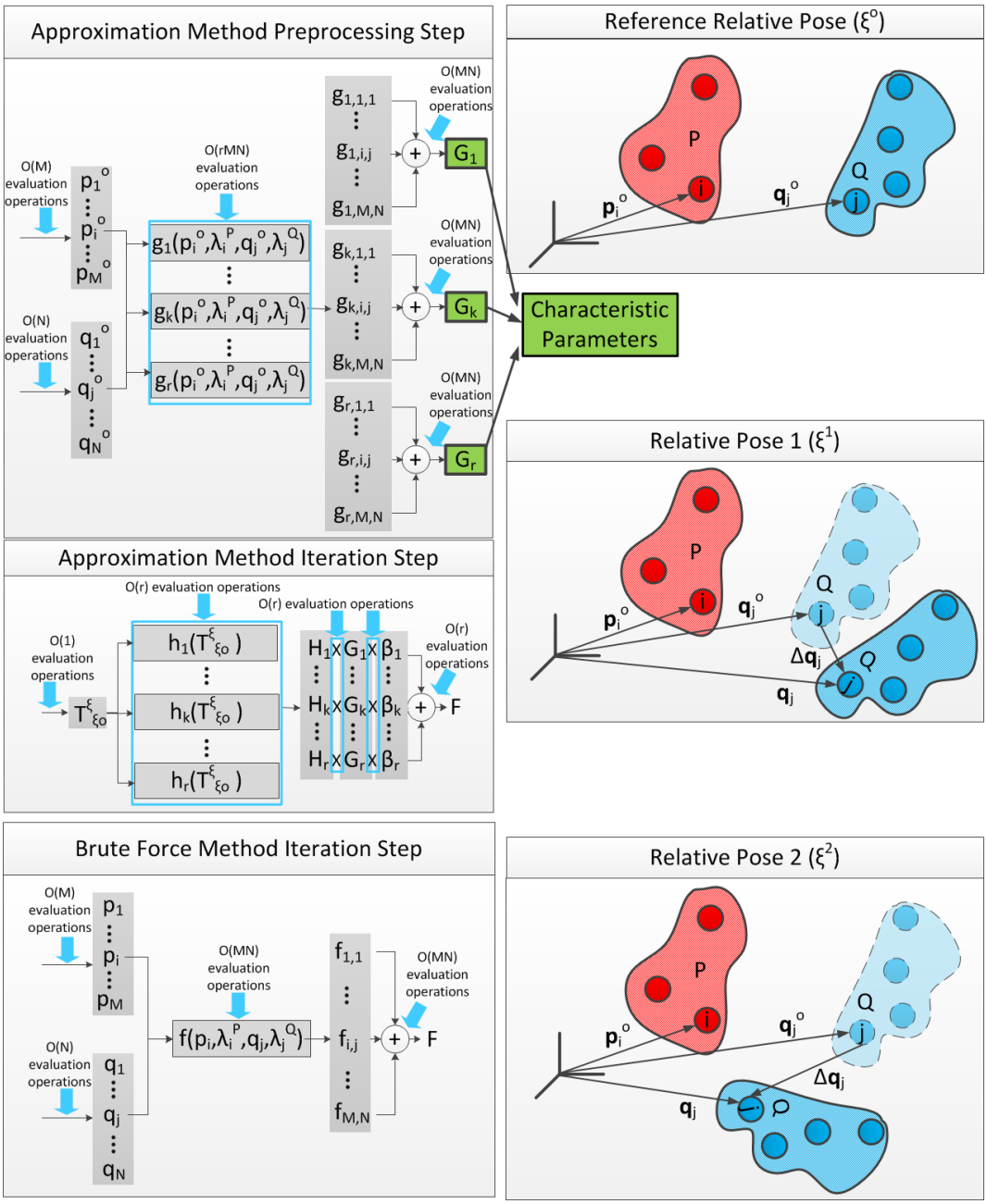


FIGURE 5.1.2: Brute force versus the proposed approximation method for computing the net interaction between two rigid bodies at multiple poses.

H of k variables into a finite sum of products of k functions of single variables, as well as, conditions for existence of special decompositions. Moreover, [66] reviews the recent advances on the use of separated representations.

Overlooking the existing approximate decomposition methods, two observations are made: (1) each one of these methods involves some sort of data-driven training and optimization. (2) each method best fits certain types of multivariate functions.

We propose an approximate decomposition method for the prevalent distance-dependent interactions i.e., for the cases that the pairwise interaction is stated only as a function of the pairwise distance and some constant (over all poses) scalar values (e.g., electrostatic, van der Waals, gravity, spring, collision, etc.).[2] Our method, first surrogates the interaction function as a polynomial of square of distance ($\sum_{k=0}^{n} a_k (d^2)^k$), using linear regression method. This is followed by substituting d^2 in the polynomial with its equivalent, in terms of reference geometry and pose variables, and finally expanding the polynomial to get a finite sum of products. It is worthwhile noting that, choosing d^2 over d as the independent variable, is key to this method which, guarantees finite number of terms, resulted from the expansion.

Using this method, the two aforementioned steps of approximation (i.e., choosing the basis functions and adjusting the regression coefficient) are conducted automatically, with minimal data-driven training. In fact, the only training that is performed here, is the linear regression that is done to get the polynomial of d^2 which, takes only fractions of a second, even for very large data. Moreover, the presented method is an essay-to-implement technique that manifests the promise of using separated representation for computing the net interaction between rigid bodies at different poses. The

[2]Note that, the pairwise distance itself, is a multivariate function of up to 12 variables (where, 6 variables describe the coordinates of a pair of points at the reference relative pose and another 6 describe the pose parameters)

utility of the method will be shown using several test cases. It is paid attention to that, the accuracy of the polynomial approximation highly depends on the degree of the polynomial and thus the complexity of representation. We Plan to embed more efficient approximation techniques into the method, in the future stages of this work.

The proposed usage of separable functions yields a fast approximation method for the net interaction between lumped masses whose complexity and accuracy are independent of the number of particles, but rather dependent on the efficiency of the used decomposition technique. The presented polynomial method, although not optimal, but offers a straightforward and almost generic (i.e., with no restriction on the form or domain of the pairwise interaction function as long as the pairwise interaction admits polynomial regression) technique for decomposing distance-dependent interactions. Moreover, the presented approach facilitates the design process of desired behaviours in a system of rigid bodies, by giving the designer the option to manipulate the characteristic values. We demonstrate that, the standard deviation of the error for the net interaction is linearly (in terms of number of particles) proportional to the regression error, if the regression errors are from a normal distribution. In the Results section, we show that as the number of particles and the number of poses which interaction must be evaluated at go up, the computational superiority of the proposed method over the brute force (or other methods whose complexities are dependent on the number of particles) can be observed more vividly. The proposed method particularly finds application in molecular simulations where, the interplay between molecular domains must be studied, as well as molecular design where, molecular domain systems must be designed with certain desired behaviours.

5.2 METHODS

In this section, we will first show how separated representation can be used for approximating the net interaction function. Then, the polynomial regression/expansion method for approximating distance dependent functions is discussed. The section is concluded with an error analysis of the proposed approximation method.

5.2.1 Separated Variable Representation

Let $\{\vec{p}_i^{\,o}|i \leq M\}$ and $\{\vec{q}_i^{\,o}|i \leq N\}$ represent the coordinates of individual particles in rigid bodies P and Q respectively, at some reference relative pose ξ^o of the two rigid bodies. Without the loss of generality, we assume body P to be fixated in the configuration space (i.e., the reference frame to be attached to body P), and therefore the up to 6 variables that describe the spatial configuration of Q, are also representative of the relative pose of the two bodies. More so, let $\{\lambda_i^P|i \leq M\}$ and $\{\lambda_i^Q|i \leq N\}$ reflect some pose-independent property of the individual particles that appears in the pairwise interaction function (e.g., particle charge). The interaction among particle i of P and particle j of Q at any relative pose ξ can be stated as:

$$f_{i,j} = f(\vec{p}_i, \lambda_i^P, \vec{q}_j, \lambda_j^Q) \tag{5.2.1}$$

or alternatively

$$f_{i,j} = f(\vec{p}_i^{\,o}, \lambda_i^P, \vec{q}_j^{\,o}, \lambda_j^Q, T_{\xi^o}^{\xi}) \tag{5.2.2}$$

where $T^{\xi}_{\xi^o}$ is the transformation operator that takes ξ^o to ξ. Assume that the pairwise interaction can be approximated by sparable functions in the following form:

$$f_{i,j} \approx \sum_{k=1}^{r} \beta_k g_k(\vec{p}_i^{\,o}, \lambda_i^P, \vec{q}_j^{\,o}, \lambda_j^Q) h_k(T^{\xi}_{\xi^o}) \tag{5.2.3}$$

where r is the separation rank [17]. Then the net interaction between the two rigid bodies can be expressed as:

$$F_{PQ} = \sum_{i=1}^{M}\sum_{j=1}^{N} f_{i,j} \approx \sum_{k=1}^{r} \big[\sum_{i=1}^{M}\sum_{j=1}^{N} \beta_k g_k(\vec{p}_i^{\,o}, \lambda_i^P, \vec{q}_j^{\,o}, \lambda_j^Q) \big] h_k(T^{\xi}_{\xi^o}) \tag{5.2.4}$$

Defining interaction characteristic constant $C_k = \sum_{i=1}^{M}\sum_{j=1}^{N} \beta_k g_k(\vec{p}_i, \lambda_i^P, \vec{q}_j, \lambda_j^Q)$, we have

$$F_{PQ} \approx \sum_{k=1}^{r} C_k h_k(T^{\xi}_{\xi^o}) \tag{5.2.5}$$

Since interaction characteristic constants are independent of pose, they can be computed once and be inserted into Eq. (5.2.5). Then for finding the net interaction at any arbitrary pose, $O(r)$ different terms must be evaluated (5.1.2).

5.2.2 Polynomial Regression/Expansion

It was shown in the previous section that, if one can get a separated representation of the pairwise interaction, they can use it towards formulating the net interaction between two lumped masses at different poses and defining the characteristic values. Now, we will see how this separated representation can be achieved by surrogating the interaction profile with a polynomial of square of distance and then expanding it.

Using polynomial regression, we get:

$$f(d, \lambda_i^P, \lambda_j^Q) \approx s(\lambda_i^P, \lambda_j^Q) \sum_{k=0}^{n} a_k (d^2)^k \tag{5.2.6}$$

where d is the distance between $\vec{p}_i$ and $\vec{q}_j$, s is function of λ_i^P and λ_j^Q, a_k is polynomial regression coefficient, and n is the degrees of the polynomial. In the case that the left hand side of 5.2.6 represents the magnitude of a vector function (i.e., $|\vec{f}(d, \lambda_i^P, \lambda_j^Q)|$) which is applied in the direction of the vector connecting $\vec{p}_i$ and $\vec{q}_j$ (e.g, pairwise force), approximation can take place componentwise, giving rise to a proper vector sum for evaluating the net interaction. For that, instead of the interaction magnitude, the ratio of interaction magnitude over distance, is surrogated as a polynomial of square of distance:

$$\frac{|\vec{f}(d, \lambda_i^P, \lambda_j^Q)|}{d} \approx s(\lambda_i^P, \lambda_j^Q) \sum_{k=0}^{n} a_k (d^2)^k, \tag{5.2.7}$$

from which, components of the pairwise interaction can be attained:

$$f_{i,j,l} = \frac{|\vec{f}(d, \lambda_i^P, \lambda_j^Q)|}{d} (q_{j,l} - p_{i,l}^o) \tag{5.2.8}$$

where,

$$\vec{q}_j = T_{\xi^o}^{\xi} \vec{q}_j^o \tag{5.2.9}$$

and subscript l takes values 1, 2 or 3 which, respectively, indicate x, y and z components. Also, in this method, certain additional treatments of the pairwise interaction, before conducting the summation over pairs, are allowed as long as the the polynomial form is preserved. For instance, one can approximate the pairwise moment by

manipulating the pairwise force interaction:

$$\vec{m}_{i,j} = \vec{q}_j \times \vec{f}_{i,j} = (T^{\xi}_{\xi^o} \vec{q}^{\,o}_j) \times \vec{f}_{i,j} \tag{5.2.10}$$

The right hand sides of equation (5.2.6) can expanded after the following replacement:

$$d^2 = (\vec{q}_j - \vec{p}^{\,o}_i) \cdot (\vec{q}_j - \vec{p}^{\,o}_i) = \sum_{l=1}^{3} (q_{j,l} - p^o_{i,l})^2 \tag{5.2.11}$$

and using Eq. 5.2.9 to substitute $\vec{q}_j$. In the most general case (where, all 6 pose parameters are variable), we can express the transformation as a combination of rotation by an angle θ about an axis in the direction of a unit vector $\vec{u} = (u_1 u_2 u_3)^t$ where $u_1^2 + u_2^2 + u_3^2 = 1$ and a translation vector $\vec{x} = (x_1 x_2 x_3)^t$ [3]:

$$T^{\xi}_{\xi_0} = \begin{bmatrix} cos\theta + u_1^2(1-cos\theta) & u_1u_2(1-cos\theta) - u_3 sin\theta & u_1u_3(1-cos\theta) + u_2 sin\theta & x_1 \\ u_2u_1(1-cos\theta) + u_3 sin\theta & cos\theta + u_2^2(1-cos\theta) & u_2u_3(1-cos\theta) - u_1 sin\theta & x_2 \\ u_3u_1(1-cos\theta) - u_2 sin\theta & u_3u_2(1-cos\theta) + u_1 sin\theta & cos\theta + u_3^2(1-cos\theta) & x_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Substituting this into equations (5.2.9), we have

$$\vec{q}_j = \begin{pmatrix} (cos\theta + u_1^2(1-cos\theta))q^o_{j,1} + (u_1u_2(1-cos\theta) - u_3 sin\theta)q^o_{j,2} + (u_1u_3(1-cos\theta) + u_2 sin\theta)q^o_{j,3} + x_1 \\ (u_2u_1(1-cos\theta) - u_3 sin\theta)q^o_{j,1} + (cos\theta + u_2^2(1-cos\theta))q^o_{j,2} + (u_2u_3(1-cos\theta) - u_1 sin\theta)q^o_{j,3} + x_2 \\ (u_3u_1(1-cos\theta) - u_2 sin\theta)q^o_{j,1} + (u_3u_2(1-cos\theta) + u_1 sin\theta)q^o_{j,1} + (cos\theta + u_3^2(1-cos\theta))q^o_{j,1} + x_3 \end{pmatrix}$$

[3] superscript ^{t} indicates transpose.

Consequently, substituting this in , (5.2.11) gives:

$$
\begin{aligned}
d^2 \quad = \quad & \Big(\boxed{q^o_{j,1}} \boxed{(cos\theta + u_1^2(1-cos\theta))} + \boxed{q^o_{j,2}} \boxed{(u_1u_2(1-cos\theta) - u_3 sin\theta)} + \boxed{q^o_{j,3}} \boxed{(u_1u_3(1-cos\theta) + u_2 sin\theta)} + \boxed{x_1} \Big)^2 \\
+ \quad & \Big(\boxed{q^o_{j,1}} \boxed{(u_2u_1(1-cos\theta) + u_3 sin\theta)} + \boxed{q^o_{j,2}} \boxed{(cos\theta + u_2^2(1-cos\theta))} + \boxed{q^o_{j,3}} \boxed{(u_2u_3(1-cos\theta) - u_1 sin\theta)} + \boxed{x_2} \Big)^2 \\
+ \quad & \Big(\boxed{q^o_{j,1}} \boxed{(u_3u_1(1-cos\theta) - u_2 sin\theta)} + \boxed{q^o_{j,1}} \boxed{(u_3u_2(1-cos\theta) + u_1 sin\theta)} + \boxed{q^o_{j,1}} \boxed{(cos\theta + u_3^2(1-cos\theta))} + \boxed{x_3} \Big)^2
\end{aligned}
$$

Note that d^2 is now represented as a sum of squares of separated representations. We will now show that this can turn into a separated representation.

Lemma 5.2.1. *Let function f have a separated representation. Then, f^n, where n is a non-negative integer, also has a separated representation.*

Proof. Suppose that f has the following form:

$$f = \sum_{k=1}^{r} g_k h_k \tag{5.2.12}$$

Then

$$f^n = \left(\sum_{k=1}^{r} g_k h_k \right)^n = (g_1h_1 + g_2h_2 + ... + g_kh_k + ... + g_rh_r)^n \tag{5.2.13}$$

$$f^n = \sum_{\forall(\alpha_1+...+\alpha_r=n)} C_{\alpha_1,...,\alpha_r} \prod_{k=1}^{r} {g_k}^{\alpha_k} {h_k}^{\alpha_k} = \sum_{\forall(\alpha_1+...+\alpha_r=n)} C_{\alpha_1,...,\alpha_r} \boxed{{g_1}^{\alpha_1} {g_2}^{\alpha_2} ... {g_k}^{\alpha_k} ... {g_r}^{\alpha_r}} \boxed{{h_1}^{\alpha_1} {h_2}^{\alpha_2} ... {h_k}^{\alpha_k} ... {h_r}^{\alpha_r}} \tag{5.2.14}$$

where α_1,...,α_r are non-negative integer numbers and

$$C_{\alpha_1,...,\alpha_r} = \binom{n}{\alpha_1} \binom{n-\alpha_1}{\alpha_2} ... \binom{n-(\alpha_1+\alpha_2+...+\alpha_{r-1})}{\alpha_r}$$

where $\binom{n}{k}$ is number of k-combinations from a set of n elements (Fig. 5.2.1). □

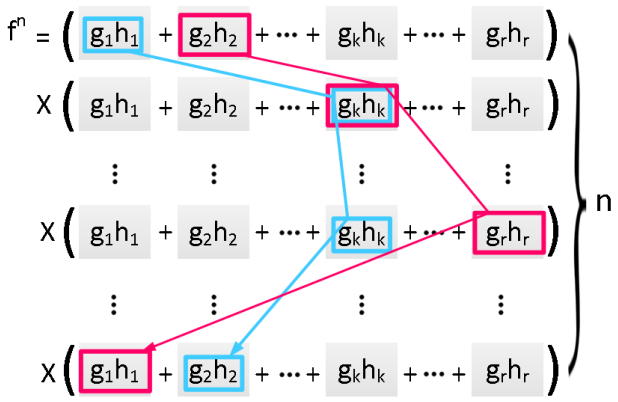


FIGURE 5.2.1: Each path results in a algebraic term that has a separated form.

From 5.2.1, it can be concluded that, d^2 and consequently, $\sum_{k=0}^{n} a_k(d^2)^k$ have separated forms.

Here as an example, we derive the separated formulation for d^2 for the case that, only two out of six pose parameters, x_1 and θ are variable (i.e., $x_2 = x_3 = u_1 = u_2 = 0$, $u_3 = 1$). The transformation will take the following form:

$$T_{\xi_0}^{\xi} = \begin{bmatrix} cos\theta & -sin\theta & 0 & x_1 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Replacing this in 5.2.9 yields:

$$\vec{q_j} = \begin{pmatrix} cos\theta q_{j,1}^o - sin\theta q_{j,2}^o + x_1 \\ sin\theta q_{j,1}^o + cos\theta q_{j,2}^o \\ q_{j,3}^o \end{pmatrix}$$

Substituting this in 5.2.11, gives:

$$d^2 = (cos\theta q^o_{j,1} - sin\theta q^o_{j,2} + x_1 - p^o_{i,1})^2 + (sin\theta q^o_{j,1} + cos\theta q^o_{j,2} - p^o_{i,2})^2 + (q^o_{j,3} - p^o_{i,3})^2 \quad (5.2.15)$$

or

$$\begin{aligned} d^2 \;=\; & \boxed{x_1^2} + \boxed{{p^o_{i,1}}^2 + {p^o_{i,2}}^2 + (q^o_{j,3} - p^o_{i,3})^2} + \boxed{-p^o_{i,1}}\boxed{2x_1} + \boxed{{q^o_{j,1}}^2 + {q^o_{j,2}}^2}\boxed{cos^2\theta} + \boxed{{q^o_{j,1}}^2 + {q^o_{j,2}}^2}\boxed{sin^2\theta} \\ + \quad & \boxed{-q^o_{j,1}p^o_{i,1} - q^o_{j,2}p^o_{i,2}}\boxed{2cos\theta} + \boxed{q^o_{j,2}p^o_{i,1} - q^o_{j,1}p^o_{i,2}}\boxed{2sin\theta} + \boxed{q^o_{j,1}}\boxed{2cos\theta x_1} + \boxed{-q^o_{j,2}}\boxed{2sin\theta x_1} \end{aligned}$$

which as can be observed, has a separated form.

5.2.3 Error Analysis

Every approximation introduces some type of error. We must see how the error of pairwise interaction contributes to the error of the net approximation. Equation 5.2.3 can be restated as

$$f_{i,j} = \sum_{k=1}^{r} \beta_k g_k(\vec{p}^o_i, \lambda^P_i, \vec{q}^o_j, \lambda^Q_j) h_k(T^{\xi}_{\xi^o}) + \epsilon_{i,j} \quad (5.2.16)$$

where, $\epsilon_{i,j}$ is the regression error. This reformulates Eq. 5.2.4 in the following form:

$$F_{PQ} = \sum_{k=1}^{r} \big[\sum_{i=1}^{M} \sum_{j=1}^{N} \beta_k g_k(\vec{p}^o_i, \lambda^P_i, \vec{q}^o_j, \lambda^Q_j) \big] h_k(T^{\xi}_{\xi^o}) + E_{net} \quad (5.2.17)$$

where $E_{net} = \sum_{i=1}^{M} \sum_{j=1}^{N} \epsilon_{i,j}$. Assuming a normal distribution for the regression error ($\epsilon_{i,j} \sim N(0, \sigma_r^2)$), we have

$$E \sim N(0, MN\sigma_r^2) = N(0, (\sqrt{MN}\sigma_r)^2) \tag{5.2.18}$$

implying that the error for the net interaction is linearly (in terms of number of particles) proportional to the regression error, if the regression errors are from a normal distribution.

5.3 RESULTS AND DISCUSSION

This section provides a few examples in which, the polynomial regression/expansion method is used to obtain a separated representation (with only one variable pose parameter) of a few types of pairwise interactions, followed by using it to evaluate the net interaction between lumped masses at different poses. We will observe that the computational time collapses from quadratic to constant yet, the resulting accuracy is reasonable.

5.3.1 Electrostatic Energy of Two Rectangular Cubes, 1D Translation

Consider the two rectangular cubes P and Q shown in Fig 5.3.1a, where point charges are distributed inside the objects. Knowing the local coordinates of each point charge inside each object, we want to evaluate the electrostatic energy of the system at a sequence of snapshots as Q travels a certain distance in x_1 direction. The pairwise

electrostatic energy for two point charges λ_i^P and λ_j^Q is

$$e_{i,j} = \frac{\lambda_i^P \lambda_i^P}{4\pi\epsilon d} \tag{5.3.1}$$

where d is the distance between the charges and ϵ is the dielectric constant. Given that the only variable pose parameter here is x_1, we have:

$$d^2 = \boxed{x_1^2} + \boxed{(q_{j,1}^o - p_{i,1}^o)^2 + {p_{i,2}^o}^2 + {p_{i,3}^o}^2} + \boxed{(q_{j,1}^o - p_{i,1}^o)}\boxed{2x_1} \tag{5.3.2}$$

We need to approximate the function $1/d$ as a polynomial of d^2. Let $a_1 = a_2 = 3$ and $a_3 = a_4 = 1$ for this example. Also, let $0 \leq x_1 \leq 5$. It can be shown that for this special case, $1 \leq d^2 \leq 82$ and thus the regression needs to be valid for this range. Choosing $n = 9$ as the degree of the polynomial regression, substituting 5.3.2 in 5.2.6 and conducting polynomial expansion results in the following expression for the pairwise energy:

$$e_{i,j} = \sum_{k=0}^{18} \beta_{k,i,j} x_1^k \tag{5.3.3}$$

where, $\beta_{k,i,j}$ is only a function of $\vec{p}_i^o$,λ_i^P,$\vec{q}_j^o$ and λ_j^Q. Figure 5.3.2 juxtaposes the values obtained for the net energy obtained from the conventional brute force method and the proposed approximation technique, as well as the time spent by each method for 3 different test cases.

Figure 5.3.3 compares the computational time of the conventional brute force method and the approximation method (preprocessing step as well as iterations for different relative poses) for different selections of two parameters, namely the number of particles in each rigid body and number of steps at which net interaction (energy

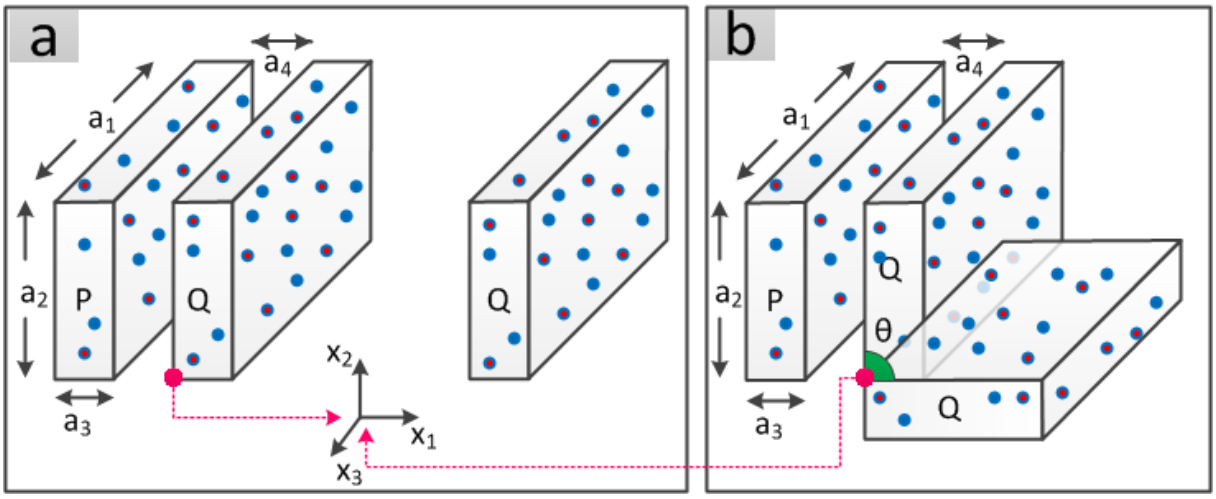


FIGURE 5.3.1: Object P is fixed. Two scenarios occur for object Q: (a) it is translated in x_1 direction; (b) it is rotated about x_3.

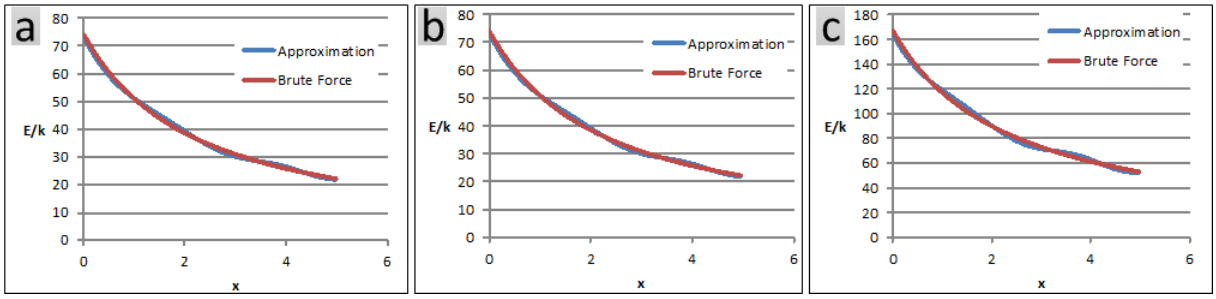


FIGURE 5.3.2: (a) Each object has 500 charged particles, randomly distributed within its confining box. The net electrostatic energy is evaluated at 500 snapshots. Conventional brute force method takes 40s of processor time compared to 6s taken by the approximation method. Average error is 1.75%. (b) Object are similar to case a. Energy is evaluated in 100 snapshots. Conventional brute force method takes 8s long, while the approximation method is accomplished in 6s. Average error is 1.75%. (c) Each object has 1000 charged particles, randomly distributed within its confining box. Energy evaluation is conducted at 500 snapshots. Conventional brute force and approximation methods last 160s and 24s respectively. Average error is 2.15%.

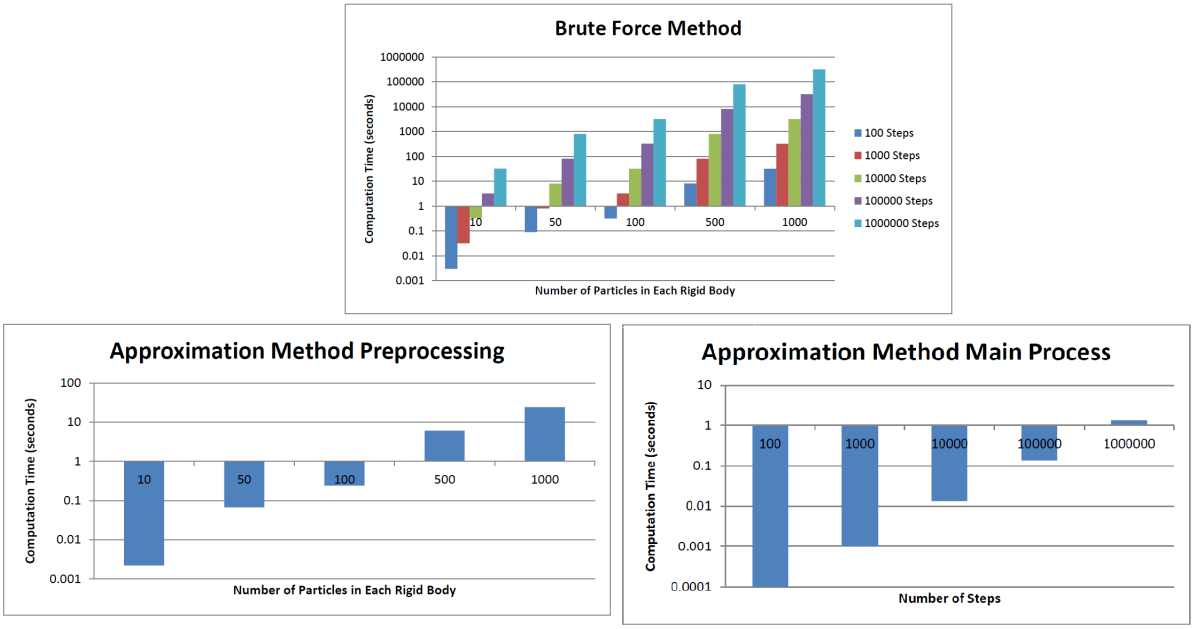


FIGURE 5.3.3: Computational time of conventional brute force method and and approximation method for different selections of number of steps and number of particles in each rigid body.

in this case) is evaluated. For the proposed method, the overall computation time is obtained by summing up the preprocessing time and that of the main process (i.e., iterations for different relative poses). Note that the number of particles only affect the former and the number of iterations (i.e., the number of poses at which the net interaction must be evaluated) only affects the latter.

5.3.2 Electrostatic Energy of Two Rectangular Cubes, $1D$ Rotation

Now, let object Q be rotated about x_3 axis. This time, given that the the only variable pose parameter is θ, we have:

$$\begin{aligned} d^2 &= \boxed{{p^o_{i,1}}^2 + {p^o_{i,2}}^2 + (q^o_{j,3} - p^o_{i,3})^2} + \boxed{{q^o_{j,1}}^2 + {q^o_{j,2}}^2}\boxed{cos^2\theta} + \boxed{{q^o_{j,1}}^2 + {q^o_{j,2}}^2}\boxed{sin^2\theta} \\ &+ \boxed{-q^o_{j,1}p^o_{i,1} - q^o_{j,2}p^o_{i,2}}\boxed{2cos\theta} + \boxed{q^o_{j,2}p^o_{i,1} - q^o_{j,1}p^o_{i,2}}\boxed{2sin\theta} \end{aligned}$$

Let the box dimensions be the same as the previous case. Also, let $-\pi/2 \leq \theta \leq 0$. Again, choosing $n = 9$ as the degree of the polynomial regression, substituting the expression for d^2 in 5.2.6, conducting polynomial expansion and noting that, $cos^2\theta = 1 - sin^2\theta$, results in the following expression for the pairwise energy:

$$e_{i,j} = \sum_{k=0}^{9} \beta_{k,i,j}(sin\theta)^k + \gamma_{k,i,j}(sin\theta)^k cos\theta \tag{5.3.4}$$

where, $\beta_{k,i,j}$ and $\gamma_{k,i,j}$ are only functions of of $\vec{p}^{\,o}_i$,λ^P_i,$\vec{q}^{\,o}_j$ and λ^Q_j. Similar to the previous case, Fig. 5.3.4 compares the performances of conventional brute force and approximation method.

5.3.3 Almost-Rigid Bodies

In several real cases, the objects under study do not remain absolutely rigid during their range of motion but rather fluctuate around a so called average rigid structure. Such phenomenon is often seen in molecular domains and usually is reflected by an index called B factor [165]. To study the effect of flexibility factor on the performance of the approximation method, we repeat the first example for 3 different levels

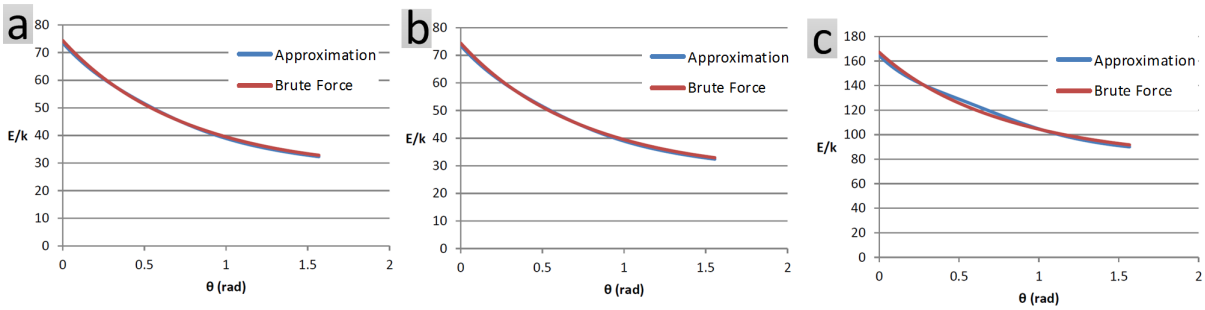


FIGURE 5.3.4: (a) Each object has 500 charged particles, randomly distributed within its confining box (same objects as case of translation example). The net electrostatic energy is evaluated at 500 snapshots. Conventional brute force method takes 40s of processor time compared to 21s taken by the approximation method. Average error is 0.99%. (b) Object are similar to case a. Energy is evaluated in 100 snapshots. Conventional brute force method takes 8s long, while the approximation method is accomplished in 21s. Average error is 0.99%. (c) Each object has 1000 charged particles, randomly distributed within its confining box. Energy evaluation is conducted at 500 snapshots. Conventional brute force and approximation methods last 159s and 84s respectively. Average error is 1.57%.

of rigidity. The brute force method computes the net interaction based on the exact coordinates of the particles at each snapshot, while the approximation method assumes a rigid body motion (Fig. 5.3.5).

5.3.4 Net Force and Net Moment Approximation

Approximations of net force and net moment are shown using another example. Let $a_1 = a_2 = a_3 = a_4 = 1$ reflect the dimensions and $0 \leq x_1 \leq 2$ for the objects of Fig. 5.3.1a. Figures 5.3.6 and 5.3.7 compare the approximation technique with the brute force method in computing different components of the net force as well as the net moment between the two objects at different snapshots.

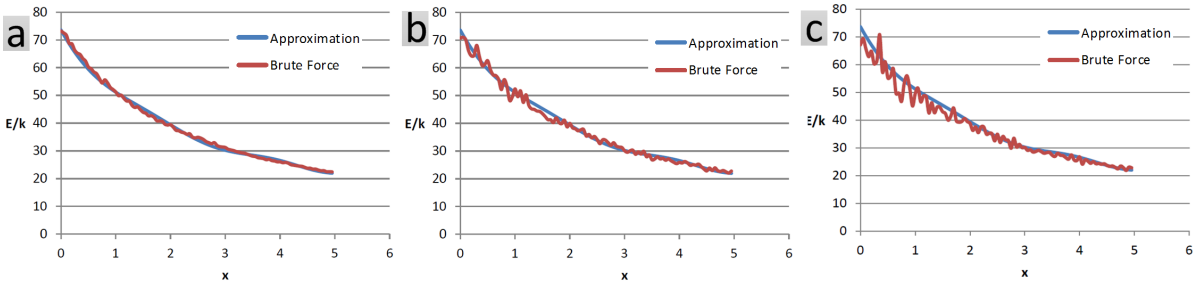


FIGURE 5.3.5: The translation example is repeated, but this time, we let particles deviate form their initial local coordinates at the reference pose, during the motion. The deviation is confined by a cube whose edge takes a certain size and is 0.1 for case a, 0.3 for case b and 0.5 for case c. Average errors for the three cases are respectively 1.77%, 2.58% and 4.05%.

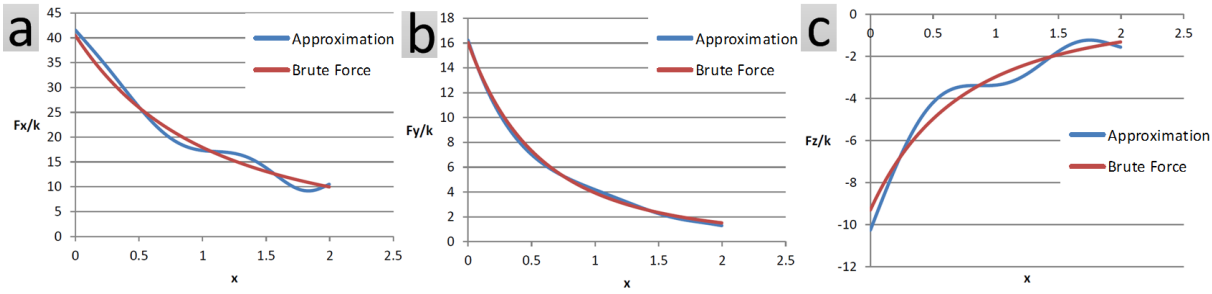


FIGURE 5.3.6: Components of the net force are computed using the approximation technique as well as the conventional brute force method. Each object has 500 charged particles, randomly distributed within its confining box. Net force evaluation is conducted at 500 snapshots. Conventional method takes 55s of processor time compared to 8s taken by the approximation method. Average error is 6.99%.

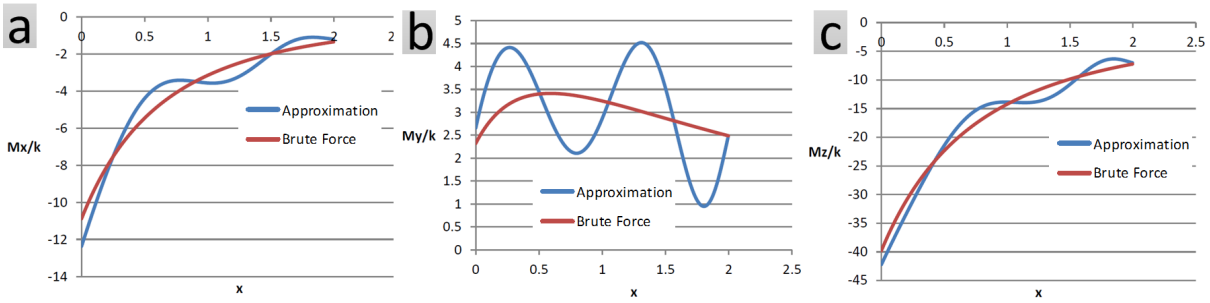


FIGURE 5.3.7: Components of the net moment are computed using the approximation technique as well as the conventional brute force method. Each object has 500 charged particles, randomly distributed within its confining box. Net moment evaluation is conducted at 500 snapshots. Conventional method takes 55s of processor time compared to 8s taken by the approximation method. Average error is 12.37%.

Electrostatic Energy of Two alpha helices, 1-D Translation

Figure 5.3.8 compares the two methods in evaluating the net interaction between two alpha helices. The first helix is fixated in the space, while the other one is translated $4\mathring{A}$ in the x direction. The electrostatic energy resulting from the partial charges on the atoms of the two objects is computed at different snapshots during the motion, both using the conventional brute force method and the suggested approximation technique.

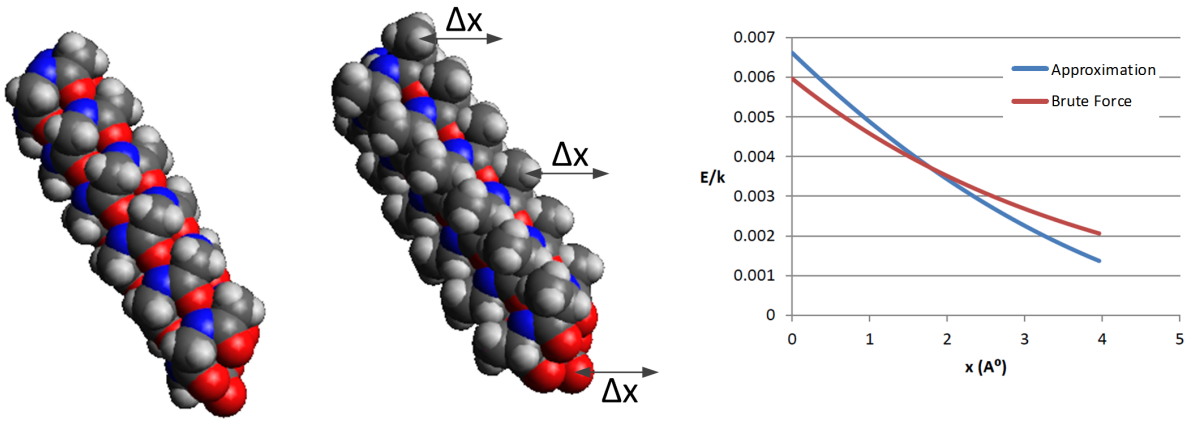


FIGURE 5.3.8: Left is an all-Glycine alpha helix with 25 residues and is fixated in space. Right is an all-Alanine alpha helix again with 25 residue and is translated $4\mathring{A}$ in the x direction. The conventional and brute force evaluations of the electrostatic energy between the two object are shown. Average error is 11.26%.

Chapter 6

CONCLUSION

Inspired by the fact that, naturally evolved protein molecules constitute only a small portion of possibilities that, can be thought for nano machines, in this dissertation, we attempted to propose a systematic strategy for design and analysis of artificial molecular machines.

In Chapter 2, we presented a novel method to synthesize machines from existing or manufacturable components, with important applications in the design of molecular machinery. The three-step process enumerates all valid topologies that satisfy prescribed mobility requirements, generates all possible linkage arrangements by assigning links to each topology from a set of available links (called the link soup), and performs a positional analysis for every linkage arrangement to produce all geometrically feasible solutions. Moreover, we determine and eliminate redundant topologies as these topologies are being generated in a computationally efficient manner. Importantly, this prompt elimination of redundant topologies significantly decreases the computational cost of the proposed design procedure. The preliminary tests suggest

that the approach produces a complete set of non-isomorphic topologies and can be used in the practical design of nano, micro as well as macro functional mechanisms that have a prescribed number of degrees of freedom. The resulting synthesis procedure is the first procedure capable of synthesizing functional linkages with prescribed mobility constructed from a soup of primitive entities. At the same time, the proposed systematic procedure outputs an ATLAS of candidate mechanisms, which can be useful in the synthesis of mechanisms for new application domains.

In Chapter 3, we applied the synthesis method developed in Chapter 2 to design irreducibly simple machines with constrained, and consequently controllable motion. Simplicity was tried to be manifested both in (a) structure, i.e. number of used components and the connections between them, for an easier manufacturing process, and (b) mobility, i.e. number of degrees of freedom (DOF) or in other words the number of independent variables needed to fully describe the motion of a machine to give rise to controllable motion. In fact we stick with the traditional tendency towards $1DOF$ machines which produce predictable and repeatable motions. The "design for function" is proposed to be broken into three smaller tasks: (1) design for mobility out of a library of building block components to reach at machines with constrained and consequently controllable motions (preferably $1DOF$) (2) devising control mechanisms to manipulate the resulting motions and (3) functionalizing the fabricated machines, to be exploited in different applications individually or as parts of an ensemble. A computational framework is developed to carry out a good portion of the aforementioned items as well as additional tasks such as compiling suitable input libraries and conducting physicals simulations. Moreover, the outputs are validated via various physical experiments.

In Chapter 4, we enhanced the previously developed KCM approach to pro-

tein folding [82, 86] originally implemented into Protofold I [83–85] which provided a promising fast alternative to the popular MD simulation and MC sampling methods by

1. modeling the protein chain as a kinematic linkage with restricted DOF to which the well-studied principles from mechanism synthesis and robotics can be readily applied; and

2. replacing the 2nd-order dynamic response with 1st-order kinetostatic integration of the equations of motion to facilitate convergence to the free energy minima.

We introduced major model and implementation improvements in Protofold II by

1. incorporating the solvation effects that characterize the hydrophobic effect, i.e., the entropic changes due to cavity formation in the aqueous solvent;

2. taking advantage from efficient auxiliary algorithms and data structures to improve the computational complexity from $O(n^2)$ to expected $O(n)$; and

3. implementing fast and relatively accurate evaluation of the SASA and its gradient for solvation energy- and force-field computation, respectively, in parallel on both CPU and GPU.

The presented enumeration algorithm for the latter provides a fast approximate method, in which the degree of accuracy is traded off with the performance by a proper choice of the sample size. We argued that the inclusion of the solvation free energy into the mix can significantly affect the folding pathway for water-soluble proteins whose folding *in vivo* is dominated by such effects. We also demonstrated

that the performance gain of the GPU-accelerated implementation scales properly with the number of atoms, achieving up to two orders of magnitude in speed-ups after memory optimization.

Protofold II has been completely rearchitectured and is evolving into a versatile analysis toolbox for studying the kinematic and structural behavior of molecular chains in protein engineering applications such as the design of nano-manipulators [157,163] among the ongoing projects. A hybrid force-field model was used, composed of

1. the AMBER model [174] (for noncovalent interactions); and
2. supplemental terms similar to the CHARMM model [177] and the GROMOS model [55] (for solvation effects) except for the probabilistic SASA estimation [180] replaced with our own surface sampling algorithm.

This model is by no means versatile enough to enable addressing the ultimate goal of arriving from sequence to 3D structure at a click of a button. Even though predicting folding of real 3D structures requires further developments, this study represents a major step toward this goal.

Despite all the breakthroughs in the field of computer algorithms, as well as the huge computational power available on even personal computers, the quadratic computational complexity associated with exact pairwise interaction evaluations remains the bottleneck of static and dynamic simulations. This has caused an ongoing search for efficient approximation methods. In chapter 5, we proposed an approximation method for the special case that the net interaction between two rigid bodies, resulting from the cumulative effect of pairwise interactions between their constituents, must be evaluated. A linear predictor, the basis functions of which have separated

forms, is used to approximate the values of a multivariate interaction function. In other words, the variables that describe the local geometries of the two rigid bodies and the ones that reflect the relative pose between them are split. This facilitates certain summation operations, when computing the net interaction. As a result, the quadratic number of interaction evaluations for each relative pose is replaced with a one-time quadratic computation of a set of characteristic parameters at a preprocessing step, plus constant number of pose function evaluations at each pose, where this constant is determined by the required accuracy of approximation as well as the efficiency of the used approximation method. We showed that the standard deviation of the error for the net interaction is linearly proportional to the regression error, if the regression errors are from a normal distribution. Our results also showed that even exploiting the simple polynomial regression/expansion method to obtain the separated representation can yield faster computation yet comparable in accuracy to the conventional brute force method. This promises even better performances if more efficient fitting algorithms are employed.

Appendix A

Peptide Chains

This appendix overviews the structural biochemistry of peptide chains. Amino acids (AAs) are composed of a central carbon atom (denoted C_α) attached to 4 chemical components; namely a carboxylate group ($-COO^-$), an amino group ($-NH_3^+$), and a hydrogen atom ($-H$), common among all types, and a variable side chain (denoted $-R$) [91]. The amino group of one AA reacts with the carboxyl group of another to form a 'peptide bond,' eliminating a water molecule. This so-called 'condensation reaction' repeats over and over again to form a 'peptide chain' [91].

As depicted in Fig. 4.1.1, the 3D structure of the peptide chain can be uniquely represented by a set of bond lengths, and two sets of angles, namely the angles between adjacent bonds that share one atom (referred to as 'bond angles'), and those describing rotation around the bonds (referred to as 'torsion angles' or 'dihedral angles'). It is reasonable to assume that the bond lengths and bond angles are constant [86], thus the dihedral angles exclusively specify the protein conformation. For a protein with m AA residues denoted by AA_i $(1 \leq i \leq m)$ numbered in order

from N-terminus to C-terminus, the 3 set of dihedral angles in the main chain are defined for $1 \leq i \leq m$ as

- the rotation angle ω_i around the backbone C−N bond that connects the residues AA_i and AA_{i+1};
- the rotation angle ϕ_i around the backbone N−C_α bond in the residue AA_i; and
- the rotation angle ψ_i around the backbone C_α−C bond in the residue AA_i.

Based on high resolution X-ray crystallographic studies, the angle ω_i is very close to either 0° (the 'cis' conformation) or 180° (the 'trans' conformation) [91], and the 6 atoms in the peptide group C_α−CO−NH−C_α are approximately coplanar, forming the so-called 'peptide plane' [86]. Due to the partial double-bond characteristic of the peptide bond C−N, the peptide groups are almost rigid, hence modeled as rigid links hinged to the preceding and following peptide groups along the main chain [86]. These planes rotate about the N−C_α and C_α−C bonds, which can be thought of as *revolute joints*. Hence the 'main chain dihedral angles' ϕ_i and ψ_i completely specify the conformation of the backbone. In addition, each side chain can be treated as a shorter linkage which can add up to 4 extra links with their associated joint angles, called 'side chain dihedral angles' ($\chi_{i,1}$ to $\chi_{i,4}$). Therefore, the whole protein chain can be modeled as an open kinematic linkage, conformation of which is fully specified by a set of main chain and side chain dihedral angles. The resulted model has a reduced number of DOF of $2m + \sum_{i=1}^{m} \mathrm{DOF}(\mathrm{R}_i)$, where the DOF of the side chain R_i is determined by the number of its side chain dihedral angles.

Appendix B

Prefix Computation

The prefix sum problem is fundamental to numerous important algorithms, and is defined as follows. Given a finite ordered sequence of elements $X = (x_1, x_2, \cdots, x_n) \in \Sigma^n$ and an arbitrary binary operator $\oplus : \Sigma \times \Sigma \to \Sigma$ that is $O(1)-$time computable and associative (i.e., $(x \oplus y) \oplus z = x \oplus (y \oplus z)$), compute another sequence $Y = (y_1, y_2, \cdots, y_n) \in \Sigma^n$ where $y_1 = x_1$, $y_2 = x_1 \oplus x_2$, $\cdots$, $y_n = x_1 \oplus x_2 \oplus \cdots \oplus x_n$; in other words $y_i = y_{i-1} \oplus x_n$ for $1 \leq i \leq n$ where y_0 is the left-identity element (i.e., $y_0 \oplus x = x$ for all $x \in \Sigma$).

It is trivial to show that the prefix sums can be computed sequentially in $O(n)$, which is optimal. In addition, there are work-optimal parallel algorithms with a total computational work of $TP = O(n)$ that carry out the prefix computation in $T = O(\log n)$ time using $P = O(n/\log n)$ CREW PRAM or in $T = O(\log n/\log\log n)$ time using $P = O(n \log\log n/\log n)$ CRCW PRAM processors with common conflict resolution [37]—see Appendix C.1 for details regarding PRAM.

Appendix C

Parallel Computing

C.1 Abstract Machines

The parallel random-access machine (PRAM) is a shared-memory abstract parallel computation model, typically assigned with the exclusive/concurrent-read (ER/CR) and exclusive/concurrent-write (EW/CW) attributes [103]. The most common attributes are CREW and CRCW, noting that multiple processors can concurrently read a memory cell but only one can write at a time to prevent race conditions. To enable concurrent writing, one needs to resolve possible conflicts with typical mechanisms such as 1) 'common' meaning that all processors attempt to write the same value); 2) 'arbitrary' meaning that one processor's attempt succeeds at random; and 3) 'priority' meaning that the processors are prioritized by a prespecified order. Note that a CREW algorithm can always run in the same (if not fewer) number of steps on a CRCW machine.

C.2 GPU SIMT Model

A typical CUDA GPU program proceeds as follows. The data is first transferred from the CPU (i.e., *host*) memory to the GPU (i.e., *device*) memory. The host application invokes the so-called kernels on the GPU with specified granularity, i.e., issuing a 1D, 2D, or 3D grid of blocks, each block being a 1D, 2D, or 3D array of threads that are sent in groups of 16 or 32 (called 'warps') to one of the streaming multiprocessors (SM). The threads within the same block can access the fast shared memory banks on the SM, and communication across blocks is done using the global memory. The computed results are transferred from the device memory back to the host memory.

There are different types of GPU memory locations, classified into two groups: 1) device (i.e., off-chip) memory including global and local memories; and 2) on-chip memory including shared memory, cache, and registers. The access latencies to the on-chip are much less (around 100× faster) than those of the off-chip memory.

Appendix D

Sifting the Conformation Space for More Efficient Sampling

It was mentioned in Chapter 2 that, for constructing plausible kinematic linkages, we will minimize an error that is associated with the overall openness of the loops, as well as the the extent of overlap between links and for that, we will sample the the hyperspace of joint variables, followed by an optimization run, starting from each sample point. In this section, we propose a technique for more efficient sampling of this hyperspace. We start with partitioning the joint variable hyperspace into a set of hypercube compartments of certain size [1]. The process is continued with sifting the resulting set, in which a quick investigation is conducted on each hypercube, resulting in two different possibilities. In one scenario, the outcome of investigation indicates that the hypercube under consideration has no chance of containing a solution point (i.e. one for which all the kinematic loops close and there are no clashes among

[1] How fine the hyperspace must be grained, depends on the characteristics of the error profile in the joint variable hyperspace, and may vary from case to case.

the links) and thus must be discarded. In another scenario, nothing specific can be concluded from the test and the hypercube may or may not carry a solution point and therefore will stay in the set. Note that, in practice, this causes a shrinkage in the sampling space and will raise the chances of finding solutions in shorter durations, by improving the sampling efficiency.

It was said that, each hypercube must be tested against a set of loop closure criteria as well as a set of overlap criteria. Nevertheless, it should be noted that each criterion involves not all but only a subset of joint variables and thus is not sensitive to value alteration of the rest. For instance, consider the partial tree shown in Fig. D.0.1 which is taken from the spanning tree of Watt topology (studied in Chapter 2). The closure status of the loop that contains vertices 1, 2 and 3, is only a function of joint variables corresponding to edges a and b. This is while, the overlap status between links 1 and 4 is only a function of joint variables corresponding to edges a and c. The impartiality of the geometric criterion to some of the joint variables, makes it indifferent to distinct hypercubes that share the same value for the joint variables, to which the criterion is sensitive. This implies that, a single test can investigate not only a single hypercube but several of them simultaneously. In Fig. D.0.1, the hypercubes with the red color are treated similarly against the overlap criterion between links 1 and 4. In addition, the hypercubes with the blue color are treated similarly against the closure criteria for the loop that contains vertices 1, 2 and 3. Also, once a hypercube is filtered out, after being examined against one criterion, the necessity for conducting the remaining tests on it, would be waived. This can be the situation for the stripy cube of Fig. D.0.1. These facts, in practice, can be exploited to speed up the process of sifting.

Now we should see, how a hypercube is actually tested against a certain criterion.

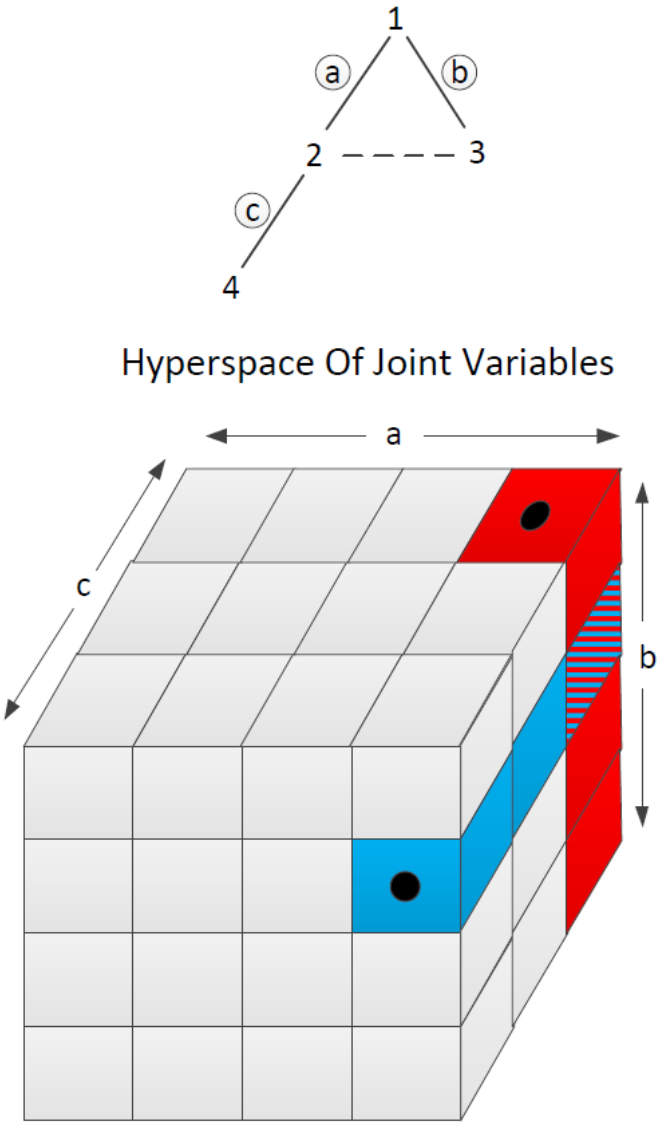


FIGURE D.0.1: Hyperspace for the the joint variables of a subtree from the Watt topology is partitioned into hypercubes. One set of cubes (red) are tested against an overlap criterion (between links 1 and 4) and another set (blue) are tested against a loop closure criterion (loop containing vertices 1, 2 and 3). The stripy cube is tested against both criteria.

The main idea is to evaluate some sort of error function, depending on the specific criterion, at the center point of the hypercube, as well as obtaining an upper bound for the amount of reduction in this error if any other point in the hypercube is selected. This upper bound is gained from a rough approximation of the workspace of the linkage as long as it picks conformations associated with points inside the hypercube. Comparing the two values can give some idea about whether the hypercube is capable of containing a solution point or not. Note that, the more precise the workspace is predicted, the higher will be the efficacy of the method in identifying and filtering out the useless hypercubes, of course at a price of more computation. Herein, we propose a rather simple approach, which provides a conservative upper bound for the error reduction. This model can be replaced with a more sophisticated one, in order to speed up the whole procedure of finding geometric solutions.

Before we can proceed to elaborate how a test is accomplished, we need to provide an accurate definition of geometric error both for joint alignment and for avoiding overlaps. The errors should be defined such that, at the favorable conformations, they grasp the zero value. First, we consider the alignment of two revolute joints. We can represent a revolute joint with a base point, and a unit vector protruded from the base point, reflecting the axis of rotation. Alignment of two revolute joints, then necessitates two incidents, namely the placement of the two joint axes on the same line, and the distance between the joint bases being equal to a prescribed value (Fig. D.0.2). This condition can be held by equivalently satisfying another set of circumstances: coincidence of two pairs of points from the two joint axes. Letting d_1 and d_2 be the distance between the first pair of points and the second pair of points, respectively, the closure error associated with the revolute joint pair can be defined as any function that grows with an increase in either of theses two distances. Here

we pick the following form: $e = d_1^2 + d_2^2$. For a pair of prismatic joints however, the situation is slightly different. To present a prismatic joint, in addition to the base point and the joint axis, a so called *guide unit vector* is needed as well, which is perpendicular to the joint axis unit vector. Here also, the two joint axes must sit on the same line. However, unlike the revolute joints, there are no restrictions on the distance between the two joint bases. Instead, there exists an additional condition to be satisfied: the two guide vectors must be parallel. In this case, the zero error must be associated with zero value for the following items: the spatial angle between the two guide vectors; the spatial angle between the two joint axes; the distance between the base of one joint and the spatial line containing the axis of the other joint (Fig D.0.3). Enforcing clash-free condition can be handled by introducing a set of distance criteria. For instance, imagine we want to avoid the geometric clash between two spheres on two separate links, at different configurations of the linkage. The clash/no-clash situation can be easily assessed by evaluating the distance between the two center points and comparing it with the summation of radii of the two spheres (Fig. D.0.4). Such way of describing and investigating the geometric hinderance is particularly favorable when we are dealing with molecular component links, composed of spherical atoms, but may be not the most efficient method, when dealing with other types of geometric links. The error can be defined as how much the distance between the two points is less than a threshold value. Note that the overall error will be defined as some weighted summation of all the individual errors.

Having different types of error defined and knowing which set of joint variables is involved in determining the value of every specific error, now we must study the variation of these errors due to alterations in the corresponding joint variables. We are specifically after finding some upper bounds for these changes, so that we can use

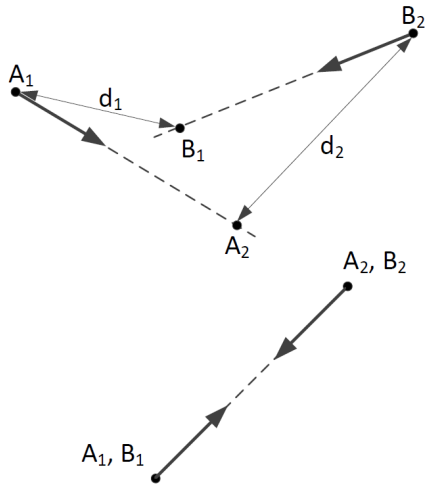


FIGURE D.0.2: Alignment of two revolute joints

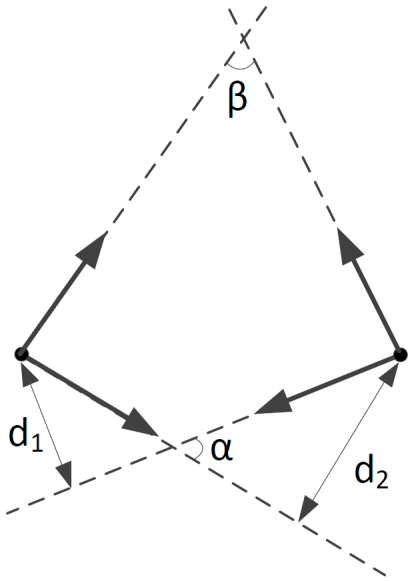


FIGURE D.0.3: Alignment of two prismatic joints. For alignment α, β, d_1 and d_2 must be zero.

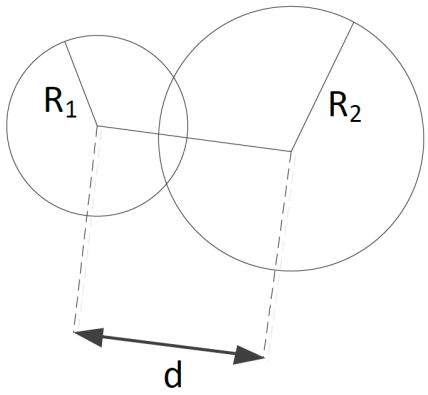


FIGURE D.0.4: Clash between a pair of spheres

them toward sifting the sampling hyperspace. Let q_i indicate the value of the element i in the maximum independent set of joint variables for a mechanism (i.e. associated with one single edge from the spanning tree). Also, let the associated hyperspace be meshed into equal-sized hypercubes such that for any point in a hypercube, we have $|q_i - q_i^c| \leq \Delta q_i$, where superscript c represents the hypercube center point, and Δq_i reflects half the size of hypercube in dimension i (Fig D.0.5). Regardless of the type of error under consideration, the value of the error at every conformation of the linkage is a function of the relative pose of a pair of links, so called *left* and *right* links. For loop closure errors, these two links are the two end of the corresponding chord and for the clash avoidance, they are the two links, geometric hinderance must be prevented for. In any case, once the two links were identified, the spanning tree will be marched upward starting from these two links until they meet at one common node, which we here call it a local root (similar to loop root for the loop closure case). Let the two legs of the spanning tree, that are marched in this process, be named the *right* chain and the *left* chain. Without the loss of generality, we assume that the local root is fixed in the space, and thus the conformation of each chain is solely determined by

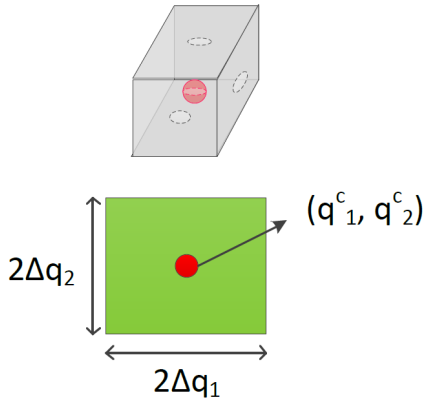


FIGURE D.0.5: Hypercube dimensions

the value of joint variables in that chain. Let $\{l_i | i \leq m\}$ and $\{r_j | j \leq n\}$ reflect the indices of the subsets of joints from the maximum independent set, present in the left and right chains, respectively (Fig D.0.6). In the following, we go through different types of error and for each, we study whether deviating the joint variables from their values at the center point can reduce the error value to a sufficient extent.

We start with the loop closure case where, the two connecting joints are revolute. As was said earlier, the closure requires coincidence of two pair of points on the two links. We consider the coincidence of one pair. Similar procedure can be performed for the other pair, as well. Let P_l and P_r be the two mates on left and right links, respectively. Their position, with respect to the local root, can be obtained by adding up vectors that starting from the joint bases of the local root, successively connect the joint bases, until they reach to the points of interest. Fig. D.0.7 shows two snapshots of the linkage: one corresponding to the conformation defined at the hypercube center point, and one at an arbitrary conformation, of course associated with a point within the hypercube. We use p for the vectors that connect the bases of two consecutive prismatic joints, and use b for others. Since the joint base of a prismatic joint is a

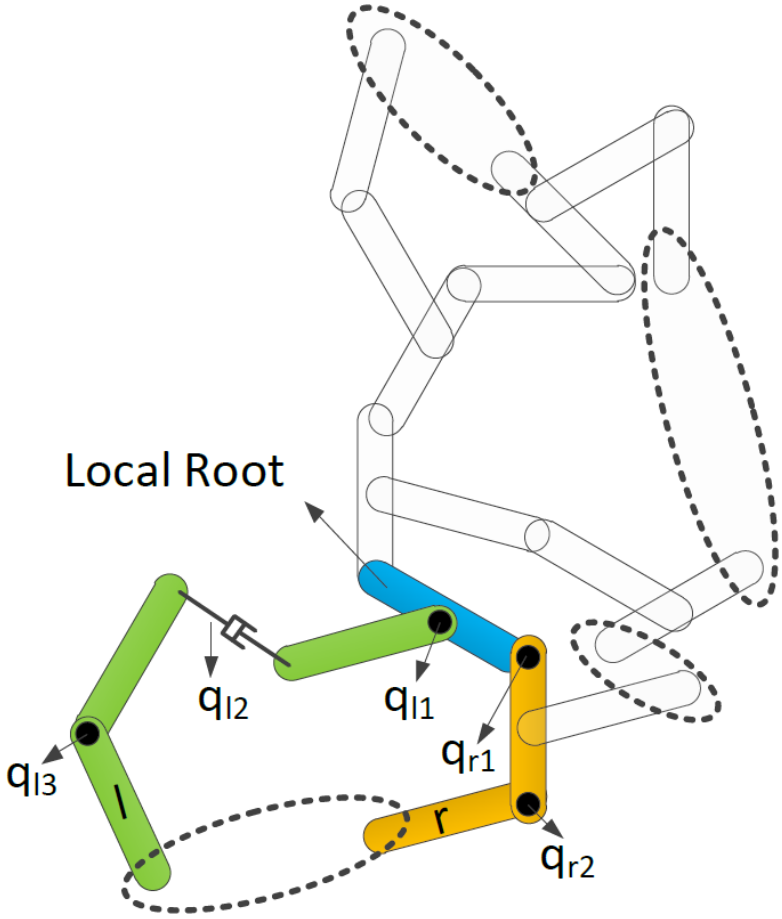


FIGURE D.0.6: Marching the spanning tree upwards, starting from the two *left* and *right* links, will result into *left* and *right* chains.

point arbitrarily selected anywhere on the joint axis, without the loss of generality and for the sake of simplicity, we assume that the p vectors vanish at the snapshot associated with the hypercube center point. Thus, we have the following:

$$A_l^c = O_l + \sum_{i=1}^{m} b_{l,i}^c \tag{D.0.1}$$

$$A_r^c = O_r + \sum_{i=1}^{n} b_{r,i}^c \tag{D.0.2}$$

$$A_l = O_l + \sum_{i=1}^{m} b_{l,i} + \sum_{i=1}^{m_p} p_{l,i} \tag{D.0.3}$$

$$A_r = O_r + \sum_{i=1}^{n} b_{r,i} + \sum_{i=1}^{n_p} p_{r,i} \tag{D.0.4}$$

In order for the two points to coincide, it is necessary that

$$A_r - A_l = 0 \tag{D.0.5}$$

or equivalently

$$(A_r - A_r^c) - (A_l - A_l^c) = A_l^c - A_r^c \tag{D.0.6}$$

Defining

$$e^c = \frac{A_l^c - A_r^c}{|A_l^c - A_r^c|} \tag{D.0.7}$$

the coincidence condition can be transformed into

$$(A_r - A_r^c) \cdot e^c - (A_l - A_l^c) \cdot e^c = |A_l^c - A_r^c| \tag{D.0.8}$$

or

$$\sum_{i=1}^{n}(b_{r,i}-b_{r,i}^{c})\cdot e^{c}+\sum_{i=1}^{n_p}p_{r,i}\cdot e^{c}$$
$$-\sum_{i=1}^{m}(b_{l,i}-b_{l,i}^{c})\cdot e^{c}-\sum_{i=1}^{m_p}p_{l,i}\cdot e^{c}=|A_{l}^{c}-A_{r}^{c}| \tag{D.0.9}$$

Whether such condition can hold or not, can be roughly investigated by attempting to obtain a conservative upper bound for the left hand side of equation (D.0.10) and then comparing it with the right hand side of the equation. If the upper bound was smaller, we will discard the hypercube from the sampling space. Otherwise, we will keep it. One upper bound for the summation appearing on the right hand side of equation (D.0.10) can be gained by adding up the upper bounds of its individual terms. This equation is essentially made up of terms from two different types. One is the projection of the change of a vector b on e^c, when the linkage conforms from the hypercube center point to another configuration. Again, an upper bound for the net change in b, which occurs due to the rotation conducted about the proceeding revolute joints, can be achieved by summing up the upper bounds for the changes in b, due to single rotations. The value of this upper bound is directly affected by relative orientation of b, e^c and the axis of rotation, u. For instance, the closer to $\pi/2$ is the angle between b and u or the angle between u and e^c, the larger would be the change in projection of b on e^c. Transforming a linkage between two fixed conformations can be done in infinitely many ways, if we alter the joint variables in different orders. Thus, different upper bounds can be attained (Fig. D.0.9). Here, we consider two scenarios: (1) if starting from the first joint prior the b vector, we move upwards in the spanning tree toward the local root, altering the joint variables on the

way (Fig D.0.9a), the angle between the joint axis and e^c is preserved at all times, and the upper bound will be $2|b|sin(\Delta q/2)|e^c \times u|$ (2) if instead, the joint variables are altered starting from the local root toward b (Fig D.0.9b), this time the angle between u and b will remain constant at all times and the upper bound would be $2|b \times u|sin(\Delta q/2)$. Among the two resulting upper bounds, the smaller one will be picked. The other term reflects the projection of a vector p on e^c. The form of this term is slightly different from the one that we just studied. However, the fact that, the value of any variable x can be stated as $x = x_0 + (x - x_0)$, we can use the math provided for b vector and obtain the upper bound, in a similar fashion.

Next, we consider the other loop closure case, in which the two chain ends are prismatic joints. Two different types of criteria must be satisfied: first one can be stated as the necessity for the spatial angle between two axes to become zero, or in other word, the necessity for the two axes to become parallel. Fig D.0.10 shows two unit vectors which are supposed to become parallel. If the vector bases remain on the center of the sphere, then the criterion is converted into the alignment of the two vectors, which in turn is equivalent to coincidence of the tips of the arrows. This new problem is quite similar to what we did for revolute joints and thus enables us to use the same mathematics here as well. The only difference is that, here, we have only one b vector but multiple joints that can transform it in the space; second criterion can be expressed as the requirement for the distance between a point and a line to become zero, or in other words, the requirement for the point to sit on the line (Fig. D.0.11). Here, we think of the two chains as one combined chain which starts from the terminal containing the vector that represents the line and ends at the other terminal. Studying the influence of all the joints between the two ends, on the position of the point, we attempt to find an upper bound for the reduction in distance.

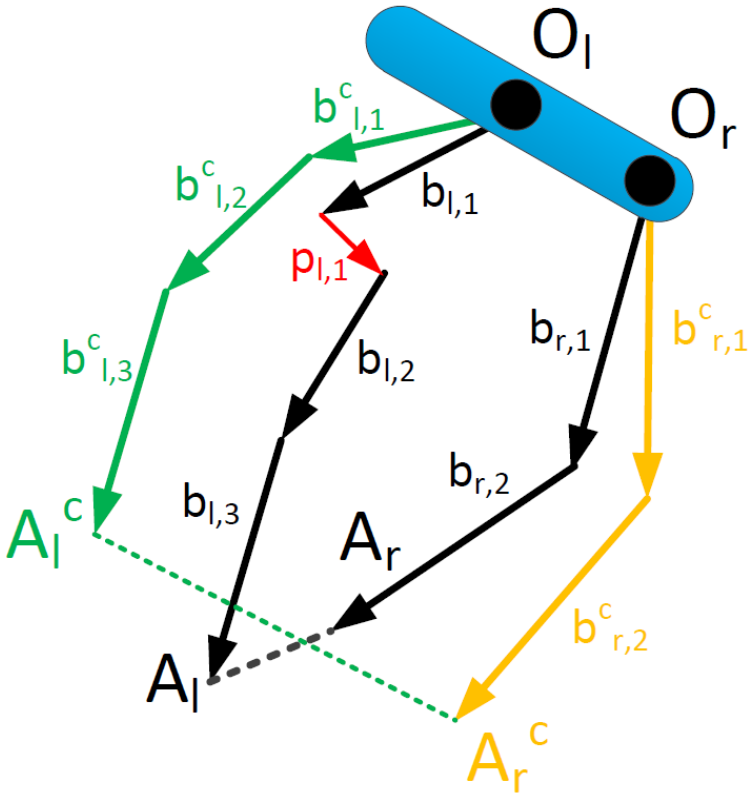


FIGURE D.0.7: The left and right chains, protruded from the local root, are shown in two snapshots. Superscript c denotes the values being evaluated at the center point of the hypercube

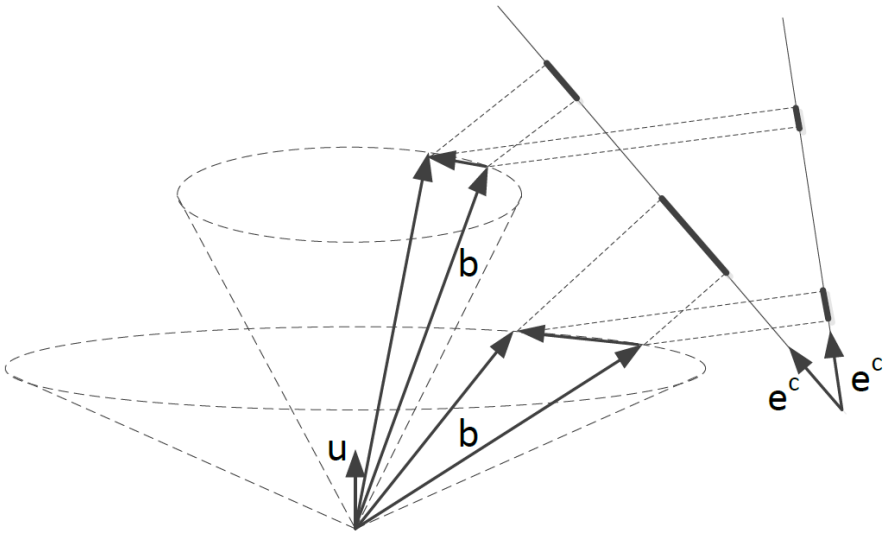


FIGURE D.0.8: The value of this upper bound is directly affected by relative orientation of b, e^c and the axis of rotation, u

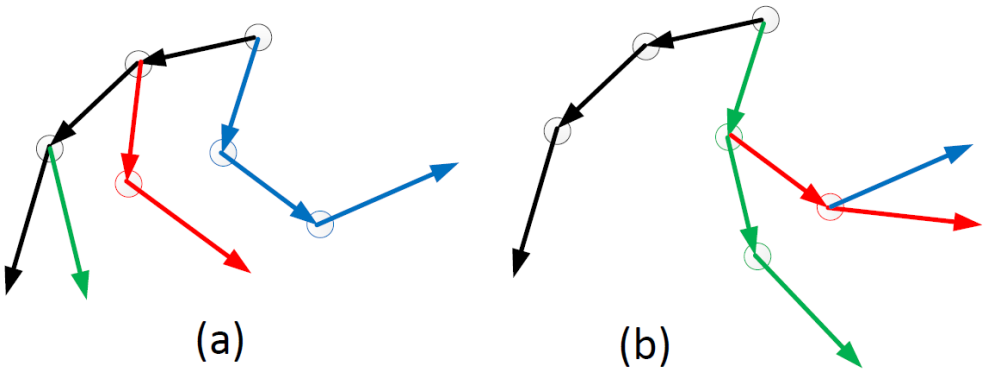


FIGURE D.0.9: Transforming a linkage between two fixed conformations can be done in infinitely many ways. Two scenarios are shown here.

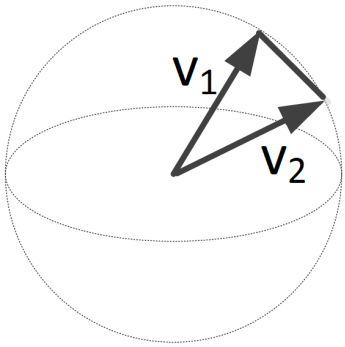


FIGURE D.0.10: Two vectors that are supposed to become parallel.

This reduction can be declared as the projection of positional change vector on the initial distance vector (Fig. D.0.11), in which the distance vector is defined as the vector that connects the point to its nearest position on the line. The net change in position is broken into its compartments, which are the positional changes associated with single joints. This is followed by obtaining a conservative upper bound by adding up the upper bounds of the compartments.

Finally, we study the case of avoiding geometric hinderance. As was mentioned earlier, in order to attain a clash-free configuration for the mechanism, a set of distance criteria must be met. Here, we examine a generic case. Let the schematics of figures D.0.6 and D.0.7 represent the dependency of the distance between an arbitrary pair of points in the mechanism on the selection of the joint variables. Defining d^t as the threshold distance between two point A_l and A_r, we want to know whether we can reach it, by picking a point from a certain hypercube. Defining d the distance between A_l and A_r, d^c the distance between A_l^c and A_r^c, $\Delta A_l = A_l - A_l^c$ and $\Delta A_r = A_r - A_r^c$, we have:

$$d^2 = |A_r - A_l|^2 = |(A_r^c - A_l^c) + (\Delta A_r - \Delta A_l)|^2 \tag{D.0.10}$$

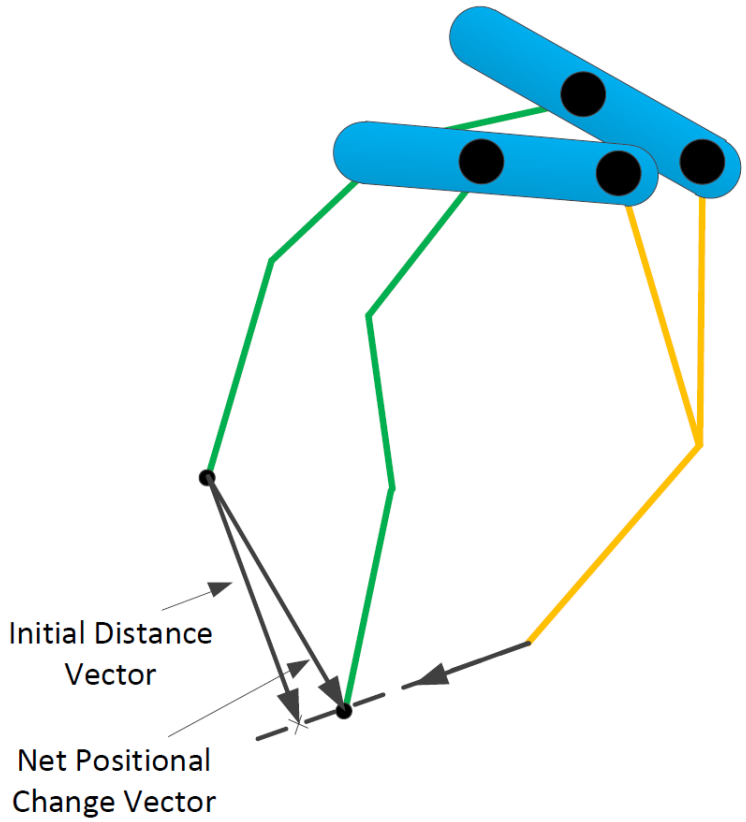


FIGURE D.0.11: The point must sit on the line, as it is required for prismatic joint closure.

Expanding the above equation will give the following:

$$d^2 = (d^c)^2 + 2(\Delta A_r - \Delta A_l) \cdot (A_r^c - A_l^c) + |\Delta A_r - \Delta A_l|^2 \tag{D.0.11}$$

Next step is to find an upper bound for d^2 and then to compare it with d^t. Computing the upper bound can be accomplished similar to the previous cases.

Bibliography

[1] R. A. Abagyan and M. Totrov, *Biased probability Monte Carlo conformational searches and electrostatic calculations for peptides and proteins*, Journal of Molecular Biology **235** (1994), no. 3, 983–1002.

[2] ———, *Ab Initio folding of peptides by the optimal-bias Monte Carlo minimization procedure*, Journal of Computational Physics **151** (1999), 402–421.

[3] J. R. Allison, K. Boguslawski, F. Fraternali, and W. F. van Gunsteren, *A refined, efficient mean solvation force model that includes the interior volume contribution*, The Journal of Physical Chemistry B **115** (2011), no. 15, 4547–4557.

[4] Andrea Altieri, Giovanni Bottari, Francois Dehez, David A Leigh, Jenny KY Wong, and Francesco Zerbetto, *Remarkable positional discrimination in bistable light-and heat-switchable hydrogen-bonded molecular shuttles*, Angewandte Chemie **115** (2003), no. 20, 2398–2402.

[5] Zhiwu An, Xiaomin Wang, Mingxi Deng, Jie Mao, and Mingxuan Li, *A nonlinear spring model for an interface between two solids*, Wave Motion **50** (2013), no. 2, 295–309.

[6] T. Ando and I. Yamato, *Free energy landscapes of two model peptides: α-helical and β-hairpin peptides explored with Brownian dynamics simulation*, Molecular Simulation **31** (2005), no. 10, 683–693.

[7] C. B. Anfinsen, *Studies on the principles that govern the folding of protein chains*, Science **181** (1973), no. 4096.

[8] Yelena A Arnautova, Jorge A Vila, Osvaldo A Martin, and Harold A Scheraga, *What can we learn by computing 13cα chemical shifts for x-ray protein models?*, Acta Crystallographica Section D: Biological Crystallography **65** (2009), no. 7, 697–703.

[9] C. Bajaj and W. Zhao, *Fast molecular solvation energetics and forces computation*, SIAM Journal on Scientific Computing **31** (2010), no. 6, 4524–4552.

[10] Rangaswami Balakrishnan and K Ranganathan, *A textbook of graph theory*, Springer Science & Business Media, 2012.

[11] Robert L Baldwin and George D Rose, *Is protein folding hierarchic? i. local structure and peptide folding*, Trends in biochemical sciences **24** (1999), no. 1, 26–33.

[12] Roberto Ballardini, Vincenzo Balzani, Alberto Credi, Maria Teresa Gandolfi, and Margherita Venturi, *Artificial molecular-level machines: which energy to make them work?*, Accounts of Chemical Research **34** (2001), no. 6, 445–455.

[13] Vincenzo Balzani, Alberto Credi, Françisco M Raymo, and J Fraser Stoddart, *Artificial molecular machines*, Angewandte Chemie International Edition **39** (2000), no. 19, 3348–3391.

[14] D. Beeman, *Some multistep methods for use in molecular dynamics calculations*, Journal of Computational Physics **20** (1976), no. 2, 130–139.

[15] M. Behandish, P. Tavousi, H. T. Ilies, and K. Kazerounian, *GPU-accelerated parallel computation of free energy for kinetostatic protein folding simulation*, Proccedings of the 2013 ASME International Design and Engineering Technical Conferences and Computer and Information in Engineering Conference (IDETC/CIE'2013), no. DETC2013-12675, American Society of Mechanical Engineers (ASME), 2013, pp. V02AT02A009:1–12.

[16] Andrea Bencini, Antonio Bianchi, Carlos Lodeiro, Andrea Masotti, A Jorge Parola, Fernando Pina, J Seixas de Melo, and Barbara Valtancoli, *A novel fluorescent chemosensor exhibiting exciplex emission. an example of an elementary molecular machine driven by ph and by light*, Chemical Communications (2000), no. 17, 1639–1640.

[17] Gregory Beylkin, Jochen Garcke, and Martin J Mohlenkamp, *Multivariate regression and machine learning with sums of separable functions*, SIAM Journal on Scientific Computing **31** (2009), no. 3, 1840–1857.

[18] R. Bonneau and D. Baker, *Ab Initio protein structure prediction: Progress and prospects*, Annual Review of Biophysics and Biomolecular Structure **30** (2001), no. 1, 173–189.

[19] J.U. Bowie, R. Luthy, and D. Eisenberg, *A method to identify protein sequences that fold into a known three-dimensional structure*, Science **253** (1991), no. 5016, 164–170.

[20] P. Bradley, K. M. S. Misura, and D. Baker, *Toward high-resolution de novo structure prediction for small proteins*, Science **309** (2005), no. 5742, 1868–1871.

[21] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, S. Swaminathan, and M. Karplus, *CHARMM: A program for macromolecular energy, minimization, and dynamics calculations*, Journal of Computational Chemistry **4** (2004), no. 2, 187–217.

[22] Wesley R Browne and Ben L Feringa, *Making molecular machines work*, Nature nanotechnology **1** (2006), no. 1, 25–35.

[23] ZIMEI Bu and DJ Callaway, *Proteins move! protein dynamics and long-range allostery in cell signaling*, Adv Protein Chem Struct Biol **83** (2011), 163–221.

[24] Horst Bunke, *Recent developments in graph matching*, Pattern Recognition, 2000. Proceedings. 15th International Conference on, vol. 2, IEEE, 2000, pp. 117–124.

[25] Glenn L Butterfoss and Brian Kuhlman, *Computer-based design of novel protein structures*, Annu. Rev. Biophys. Biomol. Struct. **35** (2006), 49–65.

[26] Martin Čadek and Jaromír Šimša, *Decomposable functions of several variables*, Aequationes Mathematicae **40** (1990), no. 1, 8–25.

[27] Adrian A Canutescu and Roland L Dunbrack, *Cyclic coordinate descent: A robotics algorithm for protein loop closure*, Protein science **12** (2003), no. 5, 963–972.

[28] J. M. Carr and D. J. Wales, *Global optimization and folding pathways of selected α-helical proteins*, Journal of Chemical Physics **123** (2005), no. 23, 234901.

[29] Benoit Champin, Pierre Mobian, and Jean-Pierre Sauvage, *Transition metal complexes as molecular machine prototypes*, Chemical Society Reviews **36** (2007), no. 2, 358–366.

[30] V. B. Chen, W. B. Arendall, J. J. Headd, D. A. Keedy, R. M. Immormino, G. J. Kapral, L. W. Murray, J. S. Richardson, and D. C. Richardson, *MolProbity: All-atom structure validation for macromolecular crystallography*, Acta Crystallographica Section D: Biological Crystallography **66** (2009), no. 1, 12–21.

[31] Kwan Wu Chin, Brian R Von Konsky, and Andrew Marriott, *Closed-form and generalized inverse kinematics solutions for the analysis of human motion*, Engineering in Medicine and Biology Society, 1997. Proceedings of the 19th Annual International Conference of the IEEE, vol. 5, IEEE, 1997, pp. 1911–1914.

[32] Gregory S Chirikjian, Kazem Kazerounian, and Constantinos Mavroidis, *Analysis and design of protein based nanodevices: Challenges and opportunities in mechanical design*, Journal of Mechanical Design **127** (2005), no. 4, 695–698.

[33] Brian Choi, Giovanni Zocchi, Yim Wu, Sum Chan, and L Jeanne Perry, *Allosteric control through mechanical tension*, Physical review letters **95** (2005), no. 7, 078102.

[34] C. Chothia and A. M. Lesk, *The relation between the divergence of sequence and structure in proteins*, The EMBO journal **5** (1986), no. 4, 823.

[35] G. Ciccotti and G. Kalibaeva, *Deterministic and stochastic algorithms for mechanical systems under constraints*, Philosophical Transactions-Royal Society

Of London Series A Mathematical Physical And Engineering Sciences **362** (2004), 1583–1594.

[36] Suzanne Clancy and William Brown, *Translation: Dna to mrna to protein*, Nature Education **1** (2008), no. 1, 101.

[37] R. Cole and U. Vishkin, *Faster optimal parallel prefix sums and list ranking*, Information and Computation **81** (1989), no. 3, 334–352.

[38] W.D. Cornell, P. Cieplak, C.I. Bayly, I.R. Gould, K.M. Merz, D.M. Ferguson, D.C. Spellmeyer, T. Fox, J.W. Caldwell, and P.A. Kollman, *A second generation force field for the simulation of proteins, nucleic acids, and organic molecules*, Journal of the American Chemical Society **117** (1995), no. 19, 5179–5197.

[39] David J Craik, *Seamless proteins tie up their loose ends*, Science **311** (2006), no. 5767, 1563–1564.

[40] Brian Curtin, *Algebraic characterizations of graph regularity conditions*, Designs, Codes and Cryptography **34** (2005), no. 2-3, 241–248.

[41] J. David and J. P. K. Doye, *Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms*, The Journal of Physical Chemistry A **101** (1997), no. 28, 5111–5116.

[42] I. W. Davis, A. Leaver-Fay, V. B. Chen, J. N. Block, G. J. Kapral, X. Wang, L. W. Murray, W. B. Arendall, J. Snoeyink, J. S. Richardson, and D. C. Richardson, *MolProbity: All-atom contacts and structure validation for proteins and nucleic acids*, Nucleic Acids Research **35** (2007), W375–W383.

[43] Ahmet Demirtas and Zahra Shahbazi, *An optimized kinematic mobility analysis of protein molecules*, ASME 2014 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, 2014, pp. V05AT08A014–V05AT08A014.

[44] Shawn M Douglas, Ido Bachelet, and George M Church, *A logic-gated nanorobot for targeted transport of molecular payloads*, Science **335** (2012), no. 6070, 831–834.

[45] David Dubbeldam, Gloria AE Oxford, Rajamani Krishna, Linda J Broadbelt, and Randall Q Snurr, *Distance and angular holonomic constraints in molecular simulations*, The Journal of chemical physics **133** (2010), no. 3, 034114.

[46] A Dubey, G Sharma, C Mavroidis, MS Tomassone, K Nikitczuk, and ML Yarmush, *Computational studies of viral protein nano-actuators*, Journal of Computational and Theoretical Nanoscience **1** (2004), no. 1, 18–28.

[47] Peter Eastman and Vijay S Pande, *Constant constraint matrix approximation: a robust, parallelizable constraint method for molecular simulations*, Journal of chemical theory and computation **6** (2010), no. 2, 434–437.

[48] P. Echenique, *Introduction to protein folding for physicists*, Contemporary Physics **48** (2007), no. 2, 81–108.

[49] M. Eilers, A. B. Patel, W. Liu, and S. O. Smith, *Comparison of helix interactions in membrane and soluble $\alpha-$bundle proteins*, Biophysical Journal **82** (2002), no. 5, 2720–2736.

[50] D. Eisenberg and A.D. McLachlan, *Solvation energy in protein folding and binding*, Nature **319** (1986), 199–203.

[51] Debra Meloy Elmegreen, Michele Kaufman, Elias Brinks, Bruce G Elmegreen, and Maria Sundin, *The interaction between spiral galaxies ic 2163 and ngc 2207. i. observations*, The Astrophysical Journal **453** (1995), 100.

[52] Pasquale Foggia, Gennaro Percannella, and Mario Vento, *Graph matching and learning in pattern recognition in the last 10 years*, International Journal of Pattern Recognition and Artificial Intelligence **28** (2014), no. 01.

[53] F. Fogolari, A. Brigo, and H. Molinari, *The Poisson–Boltzmann equation for biomolecular electrostatics: A tool for structural biology*, Journal of Molecular Recognition **15** (2002), no. 6, 377–392.

[54] Arthur E Frankel, BL Powell, NS Duesbery, GF Vande Woude, and SH Leppla, *Anthrax fusion protein therapy of cancer*, Current Protein and Peptide Science **3** (2002), no. 4, 399–407.

[55] F. Fraternali and W. F. van Gunsteren, *An efficient mean solvation force model for use in molecular dynamics simulations of proteins in aqueous solution*, Journal of Molecular Biology **256** (1996), no. 5, 939–948.

[56] T. Frembgen-Kesner and A. H. Elcock, *Striking effects of hydrodynamic interactions on the simulated diffusion and folding of proteins*, Journal of Chemical Theory and Computation **5** (2009), no. 2, 242–256.

[57] Menachem Fromer and Chen Yanover, *A computational framework to empower probabilistic protein design*, Bioinformatics **24** (2008), no. 13, i214–i222.

[58] ———, *Accurate prediction for atomic-level protein design and its application in diversifying the near-optimal sequence space*, Proteins: Structure, Function, and Bioinformatics **75** (2009), no. 3, 682–705.

[59] R. R. Gabdoulline and R. C. Wade, *Protein-protein association: Investigation of factors influencing association rates by Brownian dynamics simulations*, Journal of Molecular Biology **306** (2001), no. 5, 1139–1155.

[60] Mukesh V Gandhi and Brian S Thompson, *Smart materials and structures*, Springer Science & Business Media, 1992.

[61] Xudong Ge, Leah Tolosa, Jen Simpson, and Govind Rao, *Genetically engineered binding proteins as biosensors for fermentation and cell culture*, Biotechnology and bioengineering **84** (2003), no. 6, 723–731.

[62] C. W. Gear, *Numerical initial value problems in ordinary differential equations*, Prentice Hall, 1971.

[63] K. Ginalski, N. V. Grishin, A. Godzik, and L. Rychlewski, *Practical lessons from protein structure prediction*, Nucleic Acids Research **33** (2005), no. 6, 1874–1891.

[64] Chris D Godsil and WL Kocay, *Constructing graphs with pairs of pseudo-similar vertices*, Journal of Combinatorial Theory, Series B **32** (1982), no. 2, 146–155.

[65] Grigore Gogu, *Mobility of mechanisms: a critical review*, Mechanism and Machine Theory **40** (2005), no. 9, 1068–1097.

[66] David González, Amine Ammar, Francisco Chinesta, and Elías Cueto, *Recent advances on the use of separated representations*, International Journal for Numerical Methods in Engineering **81** (2010), no. 5, 637.

[67] Leslie Greengard and Vladimir Rokhlin, *A fast algorithm for particle simulations*, Journal of computational physics **73** (1987), no. 2, 325–348.

[68] F. Guarnieri and W.C. Still, *A rapidly convergent simulation method: Mixed Monte Carlo/stochastic dynamics*, Journal of Computational Chemistry **15** (2004), no. 11, 1302–1310.

[69] K. C. Gupta, *Kinematic analysis of manipulators using the zero reference position description*, The International Journal of Robotics Research **5** (1986), no. 2, 5–13.

[70] Thomas A Halgren, *Merck molecular force field. i. basis, form, scope, parameterization, and performance of mmff94*, Journal of computational chemistry **17** (1996), no. 5-6, 490–519.

[71] U. H. E. Hansmann and Y. Okamoto, *Comparative study of multicanonical and simulated annealing algorithms in the protein folding problem*, Physica A: Statistical Mechanics and its Applications **212** (1994), no. 3, 415–437.

[72] Marcus D Hanwell, Donald Ephraim Curtis, David C Lonie, Tim Vandermeersch, Eva Zurek, and Geoffrey R Hutchison, *Avogadro: An advanced semantic chemical editor, visualization, and analysis platform.*, J. Cheminformatics **4** (2012), no. 1, 17.

[73] HW Hellinga, *Rational protein design: combining theory and experiment*, Proceedings of the National Academy of Sciences **94** (1997), no. 19, 10015–10017.

[74] R. W. Hockney and J. W. Eastwood, *Computer simulation using particles*, Taylor & Francis, 1988.

[75] Roger W Hockney and James W Eastwood, *Particle-particle-particle-mesh (p3m) algorithms*, Computer simulation using particles.(CRC Press, 1988, pp. 267–304) (1988).

[76] S. Izvekov and G. A. Voth, *Multiscale coarse graining of liquid-state systems*, Journal of Chemical Physics **123** (2005), 134105.

[77] Joël Janin, Kim Henrick, John Moult, Lynn Ten Eyck, Michael JE Sternberg, Sandor Vajda, Ilya Vakser, and Shoshana J Wodak, *Capri: a critical assessment of predicted interactions*, Proteins: Structure, Function, and Bioinformatics **52** (2003), no. 1, 2–9.

[78] Woo Sung Jeon, Eunju Kim, Young Ho Ko, Ilha Hwang, Jae Wook Lee, Soo-Young Kim, Hee-Joon Kim, and Kimoon Kim, *Molecular loop lock: A redox-driven molecular machine based on a host-stabilized charge-transfer complex*, Angewandte Chemie **117** (2005), no. 1, 89–93.

[79] Woo Sung Jeon, Albina Y Ziganshina, Jae Wook Lee, Young Ho Ko, Jin-Koo Kang, Chongmok Lee, and Kimoon Kim, *A [2] pseudorotaxane-based molecular machine: Reversible formation of a molecular loop driven by electrochemical and photochemical stimuli*, Angewandte Chemie **115** (2003), no. 34, 4231–4234.

[80] W. L. Jorgensen and J. Tirado-Rives, *Potential energy functions for atomic-level simulations of water and organic and biomolecular systems*, Proceedings of the National Academy of Sciences of the United States of America **102** (2005), no. 19, 6665–6670.

[81] Euan R Kay, David A Leigh, and Francesco Zerbetto, *Synthetic molecular motors and mechanical machines*, Angewandte Chemie International Edition **46** (2007), no. 1-2, 72–191.

[82] K. Kazerounian, *From mechanisms and robotics to protein conformation and drug design*, Journal of Mechanical Design **126** (2004), no. 1, 40–45.

[83] ———, *From mechanisms and robotics to protein conformation and drug design*, Journal of Mechanical Design **126** (2004), no. 1, 40–45.

[84] K. Kazerounian, K. Latif, and C. Alvarado, *ProtoFold: Part II–a successive kineto-static compliance method for protein conformation prediction*, Proceedings of the 2004 ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE'2004), no. DETC2004-57247, American Society of Mechanical Engineers (ASME), 2004, pp. 669–677.

[85] ———, *Protofold: A successive kinetostatic compliance method for protein conformation prediction*, Journal of Mechanical Design **127** (2005), 712.

[86] ———, *Protofold: A successive kinetostatic compliance method for protein conformation prediction*, Journal of Mechanical Design **127** (2005), 712.

[87] Kazem Kazerounian, Khalid Latif, and Carlos Alvarado, *Protofold: A successive kinetostatic compliance method for protein conformation prediction*, Journal of Mechanical Design **127** (2005), no. 4, 712–717.

[88] K. Klenin, B. Strodel, D.J. Wales, and W. Wenzel, *Modelling proteins: Conformational sampling and reconstruction of folding kinetics*, Biochimica et Biophysica Acta (BBA)-Proteins and Proteomics **1814** (2011), no. 8, 977–1000.

[89] Takahiko Kojima, Taisuke Sakamoto, and Yoshihisa Matsuda, *Toward a photochemical and thermal molecular machine: Reversible ligand dissociation and binding in a ruthenium (ii)-2, 2'-bipyridine complex with tris (2-pyridylmethyl) amine*, Inorganic chemistry **43** (2004), no. 7, 2243–2245.

[90] E. Krieger, S.B. Nabuurs, and G. Vriend, *Homology modeling*, Methods of Biochemical Analysis **44** (2003), 509–524.

[91] J. Kuriyan, B. Konforti, and D. Wemmer, *The molecules of life: Physical and chemical principles*, Garland Science, Taylor & Francis Group, 2012.

[92] John Kuriyan, Boyana Konforti, and David Wemmer, *The molecules of life: Physical and chemical principles*, Garland Science, 2012.

[93] J. Kyte and R. F. Doolittle, *A simple method for displaying the hydropathic character of a protein*, Journal of Molecular Biology **157** (1982), no. 1, 105–132.

[94] B. Lee and F.M. Richards, *The interpretation of protein structures: Estimation of static accessibility*, Journal of Molecular Biology **55** (1971), no. 3, 379–400, IN3–IN4.

[95] P LeslieáDutton et al., *Intelligent design: the de novo engineering of proteins with specified functions*, Dalton Transactions (2006), no. 25, 3045–3051.

[96] Z. Li and H.A. Scheraga, *Monte Carlo-minimization approach to the multiple-minima problem in protein folding*, Proceedings of the National Academy of Sciences **84** (1987), no. 19, 6611–6615.

[97] Evgenii Mikhailovich Lifshits and Lev Petrovich Pitaevskii, *Statistical physics*, Pergamon Press, 1980.

[98] Jens P Linge, Michael Habeck, Wolfgang Rieping, and Michael Nilges, *Aria: automated noe assignment and nmr structure calculation*, Bioinformatics **19** (2003), no. 2, 315–316.

[99] R. D. Lins and R. Ferreira, *The stability of right- and left-handed alpha-helices as a function of monomer chirality*, Química Nova **29** (2006), no. 5, 997–998.

[100] Itay Lotan, Fabian Schwarzer, Dan Halperin, and Jean-Claude Latombe, *Algorithm and data structures for efficient energy maintenance during monte carlo simulation of proteins*, Journal of Computational Biology **11** (2004), no. 5, 902–932.

[101] LiaoFu Luo, *Quantum theory on protein folding*, Science China Physics, Mechanics and Astronomy **57** (2014), no. 3, 458–468.

[102] Tianzhi Luo, Krithika Mohan, Pablo A Iglesias, and Douglas N Robinson, *Molecular mechanisms of cellular mechanosensing*, Nature materials **12** (2013), no. 11, 1064–1071.

[103] Bruce M Maggs, Lesley R Matheson, and Robert E Tarjan, *Models of parallel computation: A survey and synthesis*, Proceedings of the 28th Hawaii International Conference on System Sciences, vol. 2, Institute of Electrical and Electronics Engineers (IEEE), 1995, pp. 61–70.

[104] V Magnasco, R McWeeny, and ZB Maksic, *Theoretical models of chemical bonding, part 4: Theoretical treatment of large molecules and their interactions*, 1991.

[105] Daniel J Mandell and Tanja Kortemme, *Backbone flexibility in computational protein design*, Current opinion in biotechnology **20** (2009), no. 4, 420–428.

[106] M. A. Martí-Renom, A. C. Stuart, A. Fiser, R. Sánchez, F. Melo, and A. Šali, *Comparative protein structure modeling of genes and genomes*, Annual Review of Biophysics and Biomolecular Structure **29** (2000), no. 1, 291–325.

[107] S. Y. Mashayak and D. E. Tanner, *Comparing solvent models for molecular dynamics of protein*, Technical report, University of Illinois at Urbana-Champaign, 2011.

[108] Raúl Méndez, Raphaël Leplae, Leonardo De Maria, and Shoshana J Wodak, *Assessment of blind predictions of protein–protein interactions: current status of docking methods*, Proteins: Structure, Function, and Bioinformatics **52** (2003), no. 1, 51–67.

[109] Markus Meringer, *Fast generation of regular graphs and construction of cages*, Journal of Graph Theory **30** (1999), no. 2, 137–146.

[110] Markus Meringer, H James Cleaves, and Stephen J Freeland, *Beyond terrestrial biology: Charting the chemical universe of α-amino acid structures*, Journal of chemical information and modeling **53** (2013), no. 11, 2851–2862.

[111] R Garey Michael and David S Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, WH Freeman & Co., San Francisco (1979).

[112] J. Moult, K. Fidelis, B. Rost, T. Hubbard, and A. Tramontano, *Critical Assessment of methods of protein Structure Prediction (CASP)–round 6*, Proteins: Structure, Function, and Bioinformatics **61** (2005), no. S7, 3–7.

[113] TS Mruthyunjaya, *Kinematic structure of mechanisms revisited*, Mechanism and machine theory **38** (2003), no. 4, 279–320.

[114] M. E. Muller, *A note on a method for generating points uniformly on $n-$dimensional spheres*, Communications of the ACM (CACM) **2** (1959), no. 4, 19–20.

[115] Hiroto Murakami, Atsushi Kawabuchi, Rika Matsumoto, Takeshi Ido, and Naotoshi Nakashima, *A multi-mode-driven molecular shuttle: photochemically and thermally reactive azobenzene rotaxanes*, Journal of the American Chemical Society **127** (2005), no. 45, 15891–15899.

[116] A. Nayeem, J. Vila, and H. A. Scheraga, *A comparative study of the simulated-annealing and Monte Carlo-with-minimization approaches to the minimum-energy structures of polypeptides:[met]-enkephalin*, Journal of Computational Chemistry **12** (2004), no. 5, 594–605.

[117] František Neuman, *Finite sums of products of functions in single variables*, Linear Algebra and its Applications **134** (1990), 153–164.

[118] Trung Dac Nguyen, Carolyn L Phillips, Joshua A Anderson, and Sharon C Glotzer, *Rigid body constraints realized in massively-parallel molecular dynamics on graphics processing units*, Computer Physics Communications **182** (2011), no. 11, 2307–2313.

[119] Yasuo Norikane and Nobuyuki Tamaoki, *Light-driven molecular hinge: a new molecular machine showing a light-intensity-dependent photoresponse that utilizes the trans-cis isomerization of azobenzene*, Organic letters **6** (2004), no. 15, 2595–2598.

[120] Anders Lau Olsen and Henrik Gordon Petersen, *Inverse kinematics by numerical and analytical cyclic coordinate descent*, Robotica **29** (2011), no. 04, 619–626.

[121] A. Onufriev, D.A. Case, and D. Bashford, *Effective Born radii in the generalized Born approximation: the importance of being perfect*, Journal of Computational Chemistry **23** (2002), no. 14, 1297–1304.

[122] D. J. Osguthorpe, *Ab Initio protein folding*, Current Opinion in Structural Biology **10** (2000), no. 2, 146–152.

[123] Shoichi Oshino, Yoko Funato, and Junichiro Makino, *Particle–particle particle–tree: A direct-tree hybrid scheme for collisional n-body simulations*, Publications of the Astronomical Society of Japan **63** (2011), no. 4, 881–892.

[124] Sheldon Park, Xi Yang, and Jeffery G Saven, *Advances in computational protein design*, Current opinion in structural biology **14** (2004), no. 4, 487–494.

[125] G. A. Petsko and D. Ringe, *Protein structure and function*, New Science Press, 2004.

[126] Mohammad Poursina and Kurt S Anderson, *Long-range force and moment calculations in multiresolution simulations of molecular systems*, Journal of Computational Physics **231** (2012), no. 21, 7237–7254.

[127] M.C. Prentiss, D. J. Wales, and P. G. Wolynes, *Protein structure prediction using basin-hopping*, Journal of Chemical Physics **128** (2008), no. 22, 225106.

[128] A.K. Rappe and C.J. Casewit, *Molecular mechanics across chemistry*, University Science Books, 1997.

[129] Anthony K Rappé, Carla J Casewit, KS Colwell, WA Goddard Iii, and WM Skiff, *Uff, a full periodic table force field for molecular mechanics and molecular dynamics simulations*, Journal of the American chemical society **114** (1992), no. 25, 10024–10035.

[130] Themistocles M Rassias and Jaromir Simsa, *Finite sums decompositions in mathematical analysis*, 1995.

[131] A. Ricci and G. Ciccotti, *Algorithms for Brownian dynamics*, Molecular Physics **101** (2003), no. 12, 1927–1931.

[132] T. J. Richmond, *Solvent accessible surface area and excluded volume in proteins: Analytical equations for overlapping spheres and implications for the hydrophobic effect*, Journal of Molecular Biology **178** (1984), no. 1, 63–89.

[133] A. Rojnuckarin, S. Kim, and S. Subramaniam, *Brownian dynamics simulations of protein folding: Access to milliseconds time scale and beyond*, Proceedings of the National Academy of Sciences **95** (1998), no. 8, 4288–4292.

[134] B. Roux and T. Simonson, *Implicit solvent models*, Biophysical Chemistry **78** (1999), no. 1, 1–20.

[135] Romelia Salomon-Ferrer, David A Case, and Ross C Walker, *An overview of the amber biomolecular simulation package*, Wiley Interdisciplinary Reviews: Computational Molecular Science **3** (2013), no. 2, 198–210.

[136] H.A. Scheraga, M. Khalili, and A. Liwo, *Protein-folding dynamics: Overview of molecular simulation techniques*, Annual Reveiw of Physical Chemistry **58** (2007), 57–83.

[137] Schrödinger, LLC, *The PyMOL molecular graphics system, version 1.8*, November 2015.

[138] O. Schueler-Furman, C. Wang, P. Bradley, K. Misura, and D. Baker, *Progress in modeling of protein structures and interactions*, Science **310** (2005), no. 5748, 638–642.

[139] A. Schug and W. Wenzel, *Predictive in silico all-atom folding of a four-helix protein with a free-energy model*, Journal of the American Chemical Society **126** (2004), no. 51, 16736–16737.

[140] ———, *An evolutionary strategy for all-atom folding of the 60-amino-acid bacterial ribosomal protein l20*, Biophysical Journal **90** (2006), no. 12, 4273–4280.

[141] J Anthony Semlyen, *Large ring molecules*, John Wiley & Son Ltd, 1996.

[142] Brigitte Servatius, Offer Shai, and Walter Whiteley, *Combinatorial characterization of the Assur graphs from engineering*, European Journal of Combinatorics **31** (2010), no. 4, 1091–1104.

[143] Z. Shahbazi, *Mechanical model of hydrogen bonds in protein molecules*, American Journal of Mechanical Engineering **3** (2015), no. 2, 47–54.

[144] Z. Shahbazi and A. Demirtaş, *Rigidity analysis of protein molecules*, Journal of Computing and Information Science in Engineering (JCISE) **15** (2015), no. 3, 031009:1–6.

[145] Z. Shahbazi, T. A. P. F. Pimentel, H. Ilieş, K. Kazerounian, and P. Burkhard, *A kinematic observation and conjecture for creating stable constructs of a peptide nanoparticle*, Advances in Robot Kinematics: Motion in Man and Machine (J. Lenarcic and M. M. Stanisic, eds.), Springer Netherlands, 2010, pp. 203–210.

[146] Zahra Shahbazi, Horea T Ilieş, and Kazem Kazerounian, *Hydrogen bonds and kinematic mobility of protein molecules*, Journal of Mechanisms and Robotics **2** (2010), no. 2, 021009.

[147] K. A. Sharp, A. Nicholls, R. Friedman, and B. Honig, *Extracting hydrophobic free energies from experimental data: Relationship to protein folding and theoretical models*, Biochemistry **30** (1991), no. 40, 9686–9697.

[148] K.T. Simons, C. Kooperberg, E. Huang, and D. Baker, *Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and bayesian scoring functions*, Journal of Molecular Biology **268** (1997), no. 1, 209–225.

[149] Adnan Sljoka, Offer Shai, and Walter Whiteley, *Checking mobility and decomposition of linkages via pebble game algorithm*, ASME Design Engineering Technical Conferences, 2011.

[150] Michael B Smith, *Organic synthesis, third edition*, vol. 1, Academics Press, An Imprint of Elsevier, 2011.

[151] W. C. Still, A. Tempczyk, R. C. Hawley, and T. Hendrickson, *Semianalytical treatment of solvation for molecular mechanics and dynamics*, Journal of the American Chemical Society **112** (1990), no. 16, 6127–6129.

[152] James J Storhoff and Chad A Mirkin, *Programmed materials synthesis with DNA*, Chemical reviews **99** (1999), no. 7, 1849–1862.

[153] Hai-Jun Su, Carlos Ernesto Castro, Alexander Edison Marras, and Michael Hudoba, *Design and fabrication of dna origami mechanisms and machines*, Advances in Reconfigurable Mechanisms and Robots I, Springer, 2012, pp. 487–500.

[154] María Suárez and Alfonso Jaramillo, *Challenges in the computational design of proteins*, Journal of The Royal Society Interface **6** (2009), no. Suppl 4, S477–S491.

[155] Raghavendran Subramanian and Kazem Kazerounian, *Improved molecular model of a peptide unit for proteins*, Journal of Mechanical Design (JMD) **129** (2007), no. 11, 1130–1136.

[156] W. C. Swope, H. C. Andersen, P. H. Berens, and K. R. Wilson, *A computer simulation method for the calculation of equilibrium constants for the formation*

of physical clusters of molecules: Application to small water clusters, Journal of Chemical Physics **76** (1982), no. 1, 637–649.

[157] P. Tavousi and H. T. Ilieş, *Synthesizing functional mechanisms from a link soup*, Proccedings of the 2015 ASME International Design and Engineering Technical Conferences and Computer and Information in Engineering Conference (IDETC/CIE'2015), no. DETC2015-47311, American Society of Mechanical Engineers (ASME), 2015, pp. V05CT08A044:1–14.

[158] Pouya Tavousi, Morad Behandish, Horea T Ilieş, and Kazem Kazerounian, *Protofold ii: Enhanced model and implementation for kinetostatic protein folding*, Journal of Nanotechnology in Engineering and Medicine **6** (2015), no. 3, 034601.

[159] Pouya Tavousi, Morad Behandish, Horea T Ilies, and Kazem Kazerounian, *Protofold ii: Enhanced model and implementation for kinetostatic protein folding*, Journal of Nanotechnology in Engineering and Medicine (2016).

[160] Pouya Tavousi, Morad Behandish, Kazem Kazerounian, and Horea T Ilieş, *An improved free energy formulation and implementation for kinetostatic protein folding simulation*, ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, 2013, pp. V06AT07A006–V06AT07A006.

[161] Pouya Tavousi, Kazem Kazerounian, and Horea Ilies, *Synthesizing functional mechanisms from a link soup*, ASME 2015 International Design Engineering Technical Conferences and Computers and Information in Engineering Con-

ference, American Society of Mechanical Engineers, 2015, pp. V05CT08A044–V05CT08A044.

[162] ———, *Synthesizing functional mechanisms from a link soup*, to appear in Journal of Mechanical Design (2016).

[163] ———, *Synthesizing functional mechanisms from a link soup*, to appear in Journal of Mechanical Design (2016).

[164] Edward Tobinick, *Tumour necrosis factor modulation for treatment of alzheimers disease*, CNS drugs **23** (2009), no. 9, 713–725.

[165] KN Trueblood, H-B Bürgi, H Burzlaff, JD Dunitz, CM Gramaccioli, HH Schulz, U Shmueli, and SC Abrahams, *Atomic dispacement parameter nomenclature. report of a subcommittee on atomic displacement parameter nomenclature*, Acta Crystallographica Section A: Foundations of Crystallography **52** (1996), no. 5, 770–781.

[166] Lung-Wen Tsai, *Mechanism design: enumeration of kinematic structures according to function*, CRC press, 2010.

[167] W. F. van Gunsteren, *The role of computer simulation techniques in protein engineering*, Protein Engineering **2** (1988), no. 1, 5–13.

[168] W. F. van Gunsteren, X. Daura, and A. E. Mark, *GROMOS force field*, Encyclopedia of Computational Chemistry (2002).

[169] WF van Gunsteren and HJC Berendsen, *A leap-frog algorithm for stochastic dynamics*, Molecular Simulation **1** (1988), no. 3, 173–185.

[170] A. Verma, S. M. Gopal, J. S. Oh, K. H. Lee, and W. Wenzel, *All-atom De Novo protein folding with a scalable evolutionary algorithm*, Journal of Computational Chemistry **28** (2007), no. 16, 2552–2558.

[171] Charles W Wampler, Jonathan D Hauenstein, and Andrew J Sommese, *Mechanism mobility and a local dimension test*, Mechanism and Machine Theory **46** (2011), no. 9, 1193–1206.

[172] H. Wang, C. Junghans, and K. Kremer, *Comparative atomistic and coarse-grained study of water: What do we lose by coarse-graining?*, The European Physical Journal E: Soft Matter and Biological Physics **28** (2009), no. 2, 221–229.

[173] Junmei Wang, Romain M Wolf, James W Caldwell, Peter A Kollman, and David A Case, *Development and testing of a general amber force field*, Journal of computational chemistry **25** (2004), no. 9, 1157–1174.

[174] S. J. Weiner, P. A. Kollman, D. A. Case, U. C. Singh, C. Ghio, G. Alagona, S. Profeta, and P. Weiner, *A new force field for molecular mechanical simulation of nucleic acids and proteins*, Journal of the American Chemical Society **106** (1984), no. 3, 765–784.

[175] J. Weiser, P. S. Shenkin, and W. C. Still, *Approximate atomic surfaces from linear combinations of pairwise overlaps (LCPO)*, Journal of Computational Chemistry **20** (1999), no. 2, 217–230.

[176] Chris Welman, *Inverse kinematics and geometric constraints for articulated figure manipulation*, Ph.D. thesis, Simon Fraser University, 1993.

[177] L. Wesson and D. Eisenberg, *Atomic solvation parameters applied to molecular dynamics of proteins in solution*, Protein Science **1** (1992), no. 2, 227–235.

[178] Wikipedia, *Nuclear magnetic resonance spectroscopy of proteins — wikipedia, the free encyclopedia*, 2016, [Online; accessed 23-April-2016].

[179] Dariusz Witt, *Recent developments in disulfide bond formation*, Synthesis (2008), no. 16, 2491–2509.

[180] S. J. Wodak and J. Janin, *Analytical approximation to the accessible surface area of proteins*, Proceedings of the National Academy of Sciences **77** (1980), no. 4, 1736–1740.

[181] Shoshana J Wodak, Marie De Crombrugghe, and Joël Janin, *Computer studies of interactions between macromolecules*, Progress in biophysics and molecular biology **49** (1987), no. 1, 29–63.

[182] R. Wolfenden, L. Andersson, P. M. Cullis, and C. C. B. Southgate, *Affinities of amino acid side chains for solvent water*, Biochemistry **20** (1981), no. 4, 849–855.

[183] Yoko Yamakoshi, Reto R Schlittler, James K Gimzewski, and François Diederich, *Synthesis of molecular-gripper-type dynamic receptors and stm-imaging of self-assembled monolayers on gold*, Journal of Materials Chemistry **11** (2001), no. 12, 2895–2897.

[184] Hong-Sen Yan and Yii-Wen Hwang, *Number synthesis of kinematic chains based on permutation groups*, Mathematical and Computer Modelling **13** (1990), no. 8, 29–42.

[185] A. Yershova and S. M. LaValle, *Deterministic sampling methods for spheres and so*(3), Proceedings of the 2004 IEEE International Conference on Robotics and Automation (ICRA'2014), vol. 4, Institute of Electrical and Electronics Engineers (IEEE), 2004, pp. 3974–3980.

[186] Horea Ilies Kazem Kazerounian Peter Burkhard Zahra Shahbazi, Tais A.P.F. Pimentel, *A kinematic observation and conjecture for stable construct of a peptide nanoparticle*, Advances in Robot Kinematics, Issue on Motion in Man and Machine, (J. Lenarcic & M. Stanisoc editors), ISBN 978-90-481-9261-8, Springer Science & Business Media (2010).