Exact Rotamer Optimization for Computational Protein Design

by

Eun-Jong Hong

S.M., Electrical Engineering and Computer Science
Massachusetts Institute of Technology
(2003)
B.S., Electrical Engineering
Seoul National University
(1998)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2008

© Massachusetts Institute of Technology 2008. All rights reserved.

Author ....
Department of Electrical Engineering and Computer Science
May 23, 2008

Certified by ....
Tomás Lozano-Pérez
TIBCO Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by ....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUL 0 1 2008
LIBRARIES

ARCHIVES

Exact Rotamer Optimization for Computational Protein Design

by

Eun-Jong Hong

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2008, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

The search for the global minimum energy conformation (GMEC) of protein side chains is an important computational challenge in protein structure prediction and design. Using rotamer models, the problem is formulated as a NP-hard optimization problem. Dead-end elimination (DEE) methods combined with systematic A^* search (DEE/A^*) have proven useful, but may not be strong enough as we attempt to solve protein design problems where a large number of similar rotamers is eligible and the network of interactions between residues is dense.

In this thesis, we present an exact solution method, named BroMAP (branch-and-bound rotamer optimization using MAP estimation), for such protein design problems. The design goal of BroMAP is to be able to expand smaller search trees than conventional branch-and-bound methods while performing only a moderate amount of computation in each node, thereby reducing the total running time. To achieve that, BroMAP attempts reduction of the problem size within each node through DEE and elimination by energy lower bounds from approximate *maximum-a-posteriori* (MAP) estimation. The lower bounds are also exploited in branching and subproblem selection for fast discovery of strong upper bounds. Our computational results show that BroMAP tends to be faster than DEE/A^* for large protein design cases. BroMAP also solved cases that were not solvable by DEE/A^* within the maximum allowed time, and did not incur significant disadvantage for cases where DEE/A^* performed well.

In the second part of the thesis, we explore several ways of improving the energy lower bounds by using Lagrangian relaxation. Through computational experiments, solving the dual problem derived from cyclic subgraphs, such as triplets, is shown to produce stronger lower bounds than using the tree-reweighted max-product algorithm. In the second approach, the Lagrangian relaxation is tightened through addition of violated valid inequalities. Finally, we suggest a way of computing individual lower bounds using the dual method. The preliminary results from evaluating BroMAP employing the dual bounds suggest that the use of the strengthened bounds does not in general improve the running time of BroMAP due to the longer running time of the dual method.

Thesis Supervisor: Tomás Lozano-Pérez
Title: TIBCO Professor of Computer Science and Engineering

Acknowledgments

I am most grateful to Professor Tomás Lozano-Pérez for serving as my advisor, introducing the problem and helping me to work on it consistently, providing unlimited and patient guidance in both carrying out the research and progressing in graduate school, and providing stable funding throughout the years.

I would like to thank Professor Bruce Tidor for his kind guidance in my study of the unfamiliar field of protein design and offering vast research resources and opportunities for collaboration. I also thank Professor Tommi Jaakkola for sharing his keen insights into the problem, and providing valuable suggestions and critical comments. Both Professor Tidor and Professor Jaakkola also served as my thesis readers.

I would like to thank current and past members of Tidor group, especially, Shaun Lippow for providing protein design test cases, scientific background and interpretations of the problem and computational results, Michael Altman for his DEE/A^* code and Alessandro Senes for his early advice and help.

I thank members of Learning and Intelligent Systems (LIS), especially Professor Leslie P. Kaelbling, for encouraging group discussions and providing opportunities to practice public talks. Han-Pang Chiu kindly spared time for a dry run of my talk whenever I have a presentation due. Samuel I. Davies and I shared office during my final year at M.I.T., and it was a blessing to have him close by for he was always more than willing to engage in a discussion on many issues including research and religion, to express his genuine thoughts on my work, and to offer thorough feedback on my practice talks.

I thank Amir Globerson and David Sontag for helpful discussions on MAP estimation methods.

I thank many Korean friends at M.I.T. for their presence and support through numerous ups and downs during my graduate study.

I gratefully acknowledge the Korea Foundation for Advanced Studies (KFAS) for providing partial stipend for the first five years and the full tuition and stipend for the third year of my study.

Finally, I would like to thank my loving parents and sister for their unwavering patience and support. In particular, my parents deserve more credit than I can ever adequately acknowledge. I believe their dedication to educating their children has been the main source of inspiration and motivation for me to reach this far.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Determining low-energy placements for side chains on a fixed backbone is an important problem in both protein structure prediction and protein design. A typical approach to the protein structure prediction is homology modeling [Chothia and Lesk, 1986, Ring and Cohen, 1993, Baker and Šali, 2001] followed by refinement of the model through determination of the side-chain conformations. Determining the side-chain conformation for a given backbone structure and an amino acid sequence is called "side-chain placement" and is solved through finding the minimum energy conformation. Protein design problems, also referred to as the "inverse folding problem" [Drexler, 1981, Pabo, 1983, Godzik et al., 1993], impose similar but bigger computational challenges than the side-chain placement problem. In the protein design problem, an amino acid sequence that will stably fold to the target backbone structure is to be found among all candidate amino acid sequences. Assuming the backbone structure is fixed and the scoring function for sequence candidates is known, the protein design problem is solved as a generalized side-chain placement problem, that is, by finding the minimum energy (or score) conformation of side chains, drawing from a range of amino acid types at each residue position [Hellinga and Richards, 1994, Dahiyat and Mayo, 1996].

This work addresses the problem of finding the minimum energy conformation of protein side-chains in the context of protein design applications. In particular, we focus on the combinatorial search aspect of the problem while the backbone structure and a sequence scoring function are given as input. A scoring function is often derived from either statistical relation between sequences and structures or molecular interaction energies of amino acids [Dima et al., 2000]. We assume a scoring function assigns a lower value for a more stable conformation, and such a value is often called an energy in this work. Assuming backbones are always fixed in our problem, the conformational space of a protein can be represented solely by its side chains' conformations. Although a side chain can have an infinite number of different conformations in theory, we approximate the conformational space of a side chain by a finite number of fixed conformations called rotamers. Therefore, the search for the minimum score conformation of a protein's side chains reduces to a discrete search problem.

The common theme throughout the thesis is development of exact solution methods for the minimum energy conformation search problem. To achieve the improvement in search speed, we rely on two main technical directions: search space reduction by branching and elimination, and effective but inexpensive bounding of conformational energies. The suggested solution methods are developed with the characteristics of protein design problems, and possible constraints on the system resources in mind.

Section 1.1 gives a background in proteins and protein structures. Section 1.2 reviews computational protein design and restricts the scope of protein design problems we will study. Section 1.3 formally defines the problem of minimum energy conformation search, called the global minimum energy conformation (GMEC) problem, originally for the side-chain packing problem. This becomes the problem we will investigate throughout the thesis. Section 1.4 describes how the protein design problem is cast into a side-chain packing problem. It addresses the large combinatorial size for practical protein design cases, and argues for finding exact solutions of large problems. Section 1.5 reviews previous work on the GMEC problem, including exact enumerative approaches and heuristic search methods. Section 1.6 presents a brief summary of our work and contributions. Section 1.7 outlines the rest of the thesis.

1.1 Background on proteins

A protein is an organic polymer chain made of amino acids. Each composing amino acid can be drawn from 20 different types of amino acids. Proteins function as important elements in biological organisms. They catalyze biochemical reactions, sustain cells structurally and mechanically, and participate in cell signaling and immune responses, to name a few. Such functions of proteins are highly related to the three-dimensional structures of proteins. It is believed that a protein's geometric arrangement is determined by its amino acid composition, and that such a structure generally corresponds to the minimum energy conformation of the protein.

Two adjacent amino acids in a protein form a peptide bond. Therefore, the entire sequence of amino acids form a long polypeptide backbone. Such a chain structure determined by the sequence of amino acids is called the primary structure of the protein. The part of an amino acid that forms the polypeptide backbone is called a residue, and the rest of the amino acid that branches from the backbone is called the side-chain. The types of amino acids are in fact determined by the atomic composition of side-chains.

The polypeptide chain is flexible and in response to atomic interactions packs into levels of complicated three-dimensional structures, called secondary, tertiary, and quaternary structure. Secondary structure refers to the regularly repeating local structures mediated by hydrogen bonds, such as α-helix and β-sheet. Tertiary structure is the overall geometric structure of a protein, determined by the positions of all the atoms. Quaternary structure is the structure of a protein complex consisting of more than one protein chains. The fold of the backbone together with the orientations of side-chains is also called the protein conformation.

The structure of a protein can be experimentally identified through procedures such as X-ray crystallography or NMR spectroscopy. X-ray crystallography provides a high-resolution electron density map, but no time-dependent information can be extracted. NMR spectroscopy provides constraints on interatomic distances, dihedral angles, and other orientational data. The information generated is of lower resolution than that from X-ray crystallography, but NMR spectroscopy can yield time-dependent information, which allows NMR spectroscopy to be used for protein folding studies. Despite the experimental validation they provide, both techniques can be experimentally intricate and time-consuming. They also require complicated computational interpretation of the generated data.

1.2 Computational protein design

Protein design refers to the process of identifying amino acid sequences that satisfy desired structural or biochemical properties. There are two main motivations for protein design. First, it serves the medical, engineering, or other scientific needs for new biochemical molecules by modifying or enhancing the functions of natural proteins, or by identifying novel proteins. Second, it provides a test bench for the current physical models of protein stability and folding, and our understanding of protein functions.

Protein design has been approached on several different levels. In the earliest stage, proteins were designed via human insights from manual examination of natural proteins [DeGrado et al., 1989, Richardson and Richardson, 1989]. However, these proteins were found to be less stable than natural proteins mainly due to poor side-chain packing. Subsequent efforts were mainly made in designing the hydrophobic core of proteins [Ponder and Richards, 1987, Hellinga and Richards, 1994, Desjarlais and Handel, 1995, Harbury et al., 1995, Dahiyat and Mayo, 1996, Lazar et al., 1997]. Due to the dominance of side-chain packing interactions, the problems were relatively simple both energetically and combinatorially. The past several years have seen the extension of the problem to non-core regions of proteins. As automatic design paradigms emerge using rotamer libraries [Ponder and Richards, 1987, Dunbrack and Karplus, 1993], active research has been pursued in the area of re-engineering natural proteins. In particular, one of the areas where computational protein design has proven most useful is to modify protein sequences so that the resulting structure adopt a desired conformational state. Some such attempts have resulted in new enzymes, stabilized proteins, solubilized membrane proteins, and enhanced protein-protein interfaces [Dwyer et al., 2004, Looger and Hellinga, 2001, Dahiyat et al., 1997, Korkegian et al., 2005, Chevalier et al., 2002, Kuhlman et al., 2003, Malakauskas and Mayo, 1998, Slovic et al., 2004, Lippow et al., 2007].

Full protein design is more challenging than partial redesign in that it requires more precise models for the combined effects of various forces, hydrogen bonding, and solvation effects. One of the early successful results for full protein design used a fixed backbone structure from the zinc-finger fold [Dahiyat and Mayo, 1997], and was followed by several other full sequence designs [Dantas et al., 2003, Kuhlman and Baker, 2004]. Proteins designed using backbone structures from natural proteins often resemble natural protein sequences, which has raised the questions of whether backbone structures contain memory for sequence information and whether full sequence design using natural backbone structures is a suitable benchmark for protein design methodologies.

The grand challenge of computational protein design is so-called *de novo* protein design, that constructs proteins with novel structures or biochemical properties, which do not have natural analogs. This is a hard problem in that there may not exist any amino acid sequence that folds to a given three-dimensional structure, for example. However, a novel globular protein was constructed from a protein-like structure built from fragments of secondary structures and through iterations of various procedures including energy minimization and backbone structure perturbation [Kuhlman et al., 2003]. Another example of de novo design is a single sequence that adopts two distinct folds by targeting stability in both states [Ambroggio and Kuhlman, 2006].

Other challenges in protein design include considerations of binding affinity and specificity, or enzymatic activity [Lippow and Tidor, 2007]. Instead of finding a sequence or conformation attaining the minimum energy state, multi-objective searches will be more adequate to many emerging applications.

It is often suggested that calculated energies of proteins contain significant errors, which can be attributed to several factors, such as inaccurate energy functions, undersampling of conformational space, and the fixed backbone assumption. Particularly, it was experimentally shown that mutations affect the backbone conformation [Baldwin et al., 1993, Lim et al., 1994]. The error from fixed backbone structure has been addressed by adopting flexible backbone design procedures. Flexible backbone design can be in large divided into two categories: (1) those separating sequence searches from backbone conformational searches [Larson et al., 2002, Kraemer-Pecore et al., 2003, Plecs et al., 2004, Ross et al., 2001, Shifman and Mayo, 2003], and (2) those explicitly integrates the backbone flexibility into a combined optimization procedure [Desjarlais and Handel, 1999, Kuhlman et al., 2003, Saunders and Baker, 2005].

In this work, **we only consider protein design under fixed backbone structures**. As discussed above, flexible backbone protein design can be approached by separate procedures for sequence optimization and backbone conformation search. In this case, the sequence optimization part can be solved by the fixed backbone design method while the backbone conformation search is handled by stochastic methods such as Monte Carlo simulation.

When the backbone is assumed to be fixed, only side-chains of amino acids are allowed degrees of freedom. Therefore, the search for the optimal sequence reduces to the so-called side-chain packing problem. The side-chain packing problem is originally defined for a single amino acid sequence and a backbone structure as follows:

Side-chain packing problem when the given amino acid sequence is folded onto the given backbone structure, what is the conformation (or orientation) of each side-chain?

Figure 1-1 illustrates the placement of side-chains through solving the side-chain packing problem. The side-chain packing problem is solved as an energy minimization problem. When the scoring function for protein sequence design accurately models the side-chain packing energy, it is obvious that the optimal sequence we look for will have the best side-chain packing energy. That is, the optimal sequence for the fixed backbone protein design problem can be found by solving the side-chain packing problem for each candidate sequence, and then picking the sequence with the min-

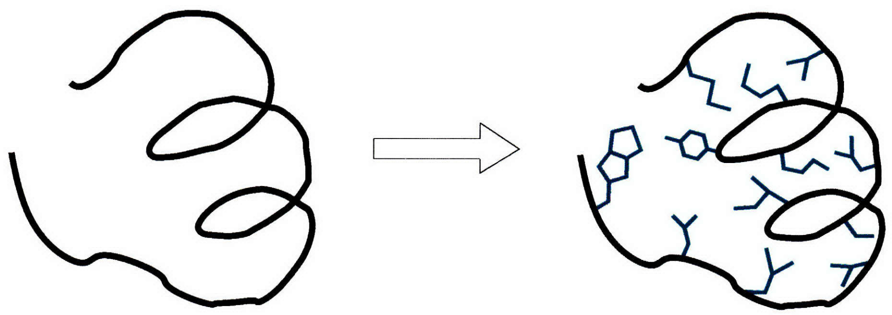


Figure 1-1: The side-chain packing problem is, given a backbone structure and an amino acid sequence, to find a correct conformation of each amino acid side-chain when the sequence is folded to the structure. The problem is often used in prediction of protein structures. It is solved by finding the conformation that minimizes the sum of interaction energies.

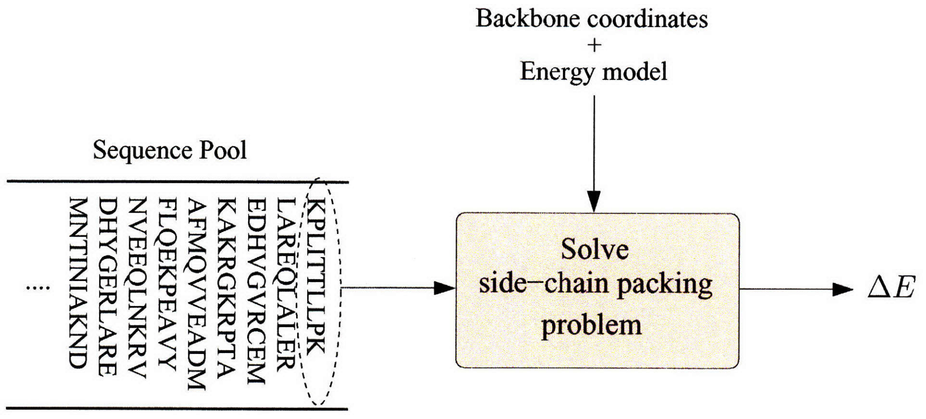


Figure 1-2: Fixed backbone protein design can be regarded as a procedure of selecting the sequence with the best side-chain packing energy. The figure illustrates a naive way of doing this, that is, by solving the side-chain packing problem for each candidate sequence.

imum packing energy. Figure 1-2 illustrates the idea of using a side-chain packing procedure for sequence selection.

1.3 Global minimum energy conformation (GMEC) search

In the previous section, we have seen the side-chain packing problem, that is, the search for the minimum energy conformation for a given amino acid sequence is an important computational challenge in computational protein design. The minimum energy conformation search is a hard problem considering each side-chain can have almost an infinite number of different conformations, corresponding to variations in its bond lengths, bond angles, and most importantly dihedral angles. In addition, time for enumerative search will be multiplied by the time for calculating energy from the physical model for each conformation.

Despite the complexity of the original minimum energy conformation search, the search space can be simplified by allowing only a finite number of fixed side-chain conformations, called rotamers [Ponder and Richards, 1987, Dunbrack and Karplus, 1993]. With the rotamer model, the energy function of a protein sequence folded onto a specific backbone template can be described in terms of [Desmet et al., 1992]:

1. the self-energy of the backbone template from the interactions within the backbone (denoted as $E_{template}$);

2. the singleton interaction energy between the backbone and rotamer conformation r at position i of the sequence (denoted as $E(i_r)$);

3. the pairwise interaction energy between rotamer conformation r at position i and rotamer conformation s at position j, $i \neq j$ (denoted as $E(i_r, j_s)$).

Then, the energy of a protein sequence of length n in a specific backbone template structure and conformation $C = \{C_1, \ldots, C_n \mid C_i \text{ is the conformation of position } i\}$ can be written in a functional form as

$$\mathcal{E}(C) = E_{template} + \sum_{i=1}^{n} E(C_i) + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E(C_i, C_j). \tag{1.1}$$

Energy terms $E(i_r)$ and $E(i_r, j_s)$ can be computed for a given backbone template and the set of allowed rotamers using coordinates of atoms and specified molecular force fields, such as AMBER [Weiner et al., 1984, 1986, Cornell et al., 1995], CHARMM [Brooks et al., 1983, Mackerell et al., 1998], MMFF [Halgren, 1996], or OPLS [Jorgensen and Tirado-Rives, 1996]. The conformation C that minimizes the energy function $\mathcal{E}(C)$ is often called the global minimum energy conformation

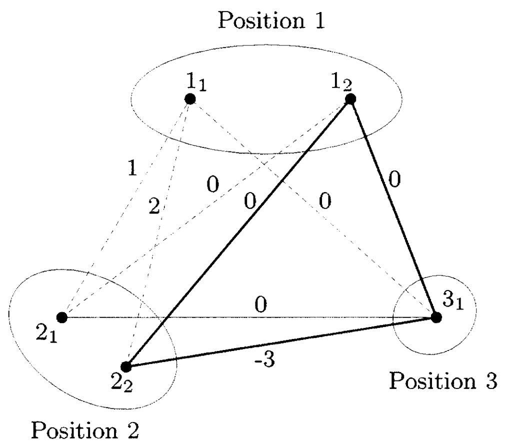


Figure 1-3: A graphical representation of a GMEC problem instance. It is easy to see that the GMEC for the problem is $(1_2, 2_2, 3_1)$ with the GMEC energy equal to -3.

(GMEC). In this work, we consider the problem of finding the GMEC when given a backbone conformation, a set of rotamers, and energy terms, and call such a problem "the GMEC problem". Note that $E_{template}$ is constant by definition and can be ignored when we minimize $\mathcal{E}(C)$.

Figure 1-3 illustrates the graphic representation of the GMEC problem that we are going to use. In the diagram, the ellipses represent the residue positions in the problem. The filled dots represent available rotamer choices at each position. The dashed line connecting rotamers at different positions represent possible interactions between pairs of rotamers. The number on the dashed line connecting rotamer i_r and j_s is the pairwise interaction energy $E(i_r j_s)$. We will often ignore the singleton interaction energies $E(i_r)$ because they can be numerically incorporated into the pairwise interaction energies. The solid lines mark the minimum energy conformation for the problem.

1.4 Protein design problem as a generalized side-chain packing problem

With the formulation of the side-chain packing problem from the previous section, the protein design problem under fixed backbone structures can be cast into a single

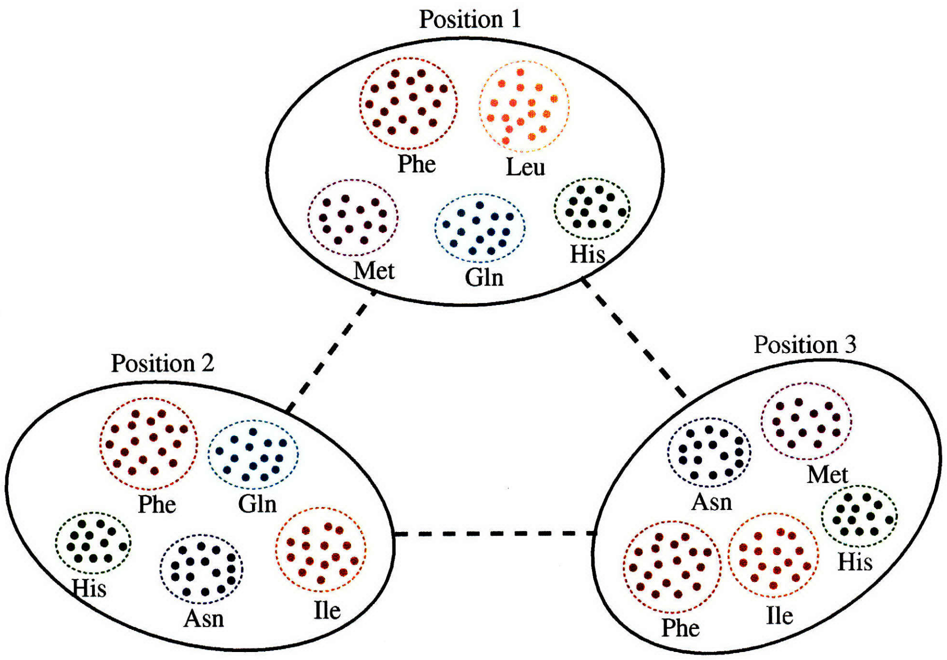


Figure 1-4: An example protein design problem cast as a large side-chain packing problem. Each design position offers rotamer choices from multiple types of amino acids.

large side-chain packing problem. Instead of solving the side-chain packing problem for each candidate sequence as suggested by Figure 1-2, by allowing rotamer choices for each sequence position to be drawn from all allowed amino acid types of that position, the resulting side-chain packing problem, if solved exactly, will identify the rotamers corresponding to the minimum energy conformation of the optimal sequence. Figure 1-4 illustrates the construction of a side-chain packing problem for solving a protein design problem.

The size of the resulting side-chain packing problem can be huge. The number of rotamers that a basic rotamer library will offer for each amino acid is 3 to the number of dihedral angles in the amino acid. Since each amino acid has 0 to 4 dihedral angles, on average $\frac{1}{5}(3^0+3^1+3^2+3^3+3^4) \approx 24$ rotamers are offered for each amino acid. Suppose we are looking for amino acids for 30 positions, the number of possible conformations that need to be evaluated becomes 24^{30}.

Considering the combinatorial size of the problem, it may look more reasonable taking heuristic approaches to the problem than trying to solve it exactly. However,

we only address exact solution methods in this work. There are two reasons for exactly solving the problem. First, the problem is also hard to obtain approximate solutions (See Appendix A.2). This implies that it is not generally expected that solutions close to the exact solution can be obtained in significantly less time than that for solving the problem exactly. Therefore, a solution that some stochastic method identifies may not be used to estimate the global minimum energy unless additional information, such as a lower bound of the minimum energy, is also provided. Empirical studies with both heuristic and exact methods to solve the GMEC problem report that the performance of the heuristic methods tends to degrade as the problem size grows [Voigt et al., 2000].

The second motivation for the exact solution of the GMEC problem is found in protein design methodology. Although computational protein design techniques have been successfully applied to finding novel biomolecules and satisfied the engineering purposes to a certain degree, the energies computed during the design process inherently contain errors from inaccurate physical models, poor sampling of the conformational space by rotamers, or the fixed backbone assumption. Despite the undesired inaccuracy that may have been introduced during the process of sequence search, it also implies that the adopted models such as scoring functions and rotamer libraries will have chances to be improved through calibration between the theoretical models and the physical results. This can be in fact a more significant benefit of studies of computational protein design in the current stage, where active research is pursued in the direction of developing protein design methodologies, than pure engineering use of protein design. However, if the GMEC problem is solved only approximately, and the solution found does not provide any information on its quality, such uncertainty limits the use of results to correct the models or design protocols. This is suggested in the protein design automation cycle [Dahiyat and Mayo, 1996] of Figure 1-5.

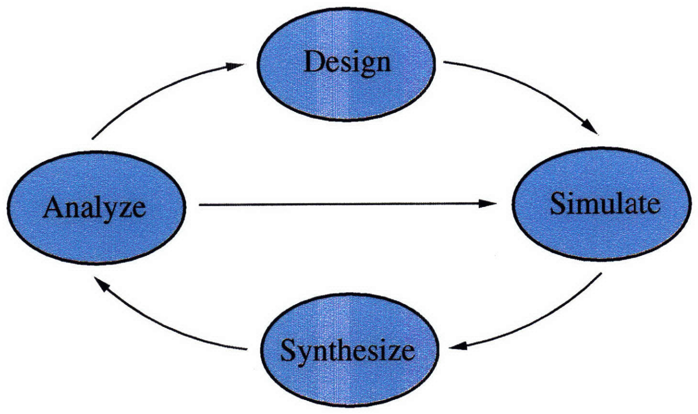


Figure 1-5: The iterative loop of protein design procedures [Dahiyat and Mayo, 1996]. In this loop, one sets up a problem with desired backbone structure and amino acids in the design step. Then, in the simulation step, the GMEC search is performed to identify a sequence that satisfies the design goal. The sequence is synthesized and characterized in the next step of synthesis. Finally, in the analysis step, one can correct the potential energy function used in the simulation by comparing the experimental data and the model, or even modify the problem setups in the design step.

1.5 Related work

In this section, we review the most widely used exact method called dead-end elimination to solve the GMEC problem for protein design, and other exact methods and approximate methods that have been employed to solve the GMEC problem.

1.5.1 Dead-end elimination

The GMEC problem is a strongly *NP*-hard optimization problem as one can readily show by reduction from the satisfiability problem (see Appendix A). Despite the theoretical hardness, one finds that many instances of the GMEC problem are easily solved by the exact method of dead-end elimination (DEE) [Desmet et al., 1992].

Dead-end elimination (DEE) identifies rotamers and combinations of rotamers that cannot be a part of the GMEC. The basic idea to find a candidate for elimination is to identify a rotamer of a residue that always imparts less favorable energy to the conformation than other rotamers in the same residue. That is, if there exist two rotamers r and t of residue i that satisfy the following inequality for every possible conformation for the other residues of the protein, then we can eliminate the rotamer i_r from the conformational search.

$$E(i_r) + \sum_{j,j \neq i} E(i_r j_s) > E(i_t) + \sum_{j,j \neq i} E(i_t j_s) \tag{1.2}$$

However, checking the above condition for every possible conformation of other residues and finding a rotamer to be eliminated is computationally impractical. The following is a weaker elimination condition but it is computationally far easier:

$$E(i_r) + \sum_{j,j \neq i} \min E(i_r j_s) > E(i_t) + \sum_{j,j \neq i} \max E(i_t j_s) \tag{1.3}$$

An extension of (1.3) for rotamer-pair elimination also exists [Desmet et al., 1992]. Such DEE conditions are applied iteratively until no more rotamers can be eliminated. When DEE sufficiently reduces the problem size, the GMEC is readily found.

Since the early work by Desmet et al. [1992] on singles and pairs elimination conditions, there have been various modifications and improvements of the DEE algorithm. Goldstein [1994] presented an improved version of the elimination inequality (1.3) as follows:

$$E(i_r) - E(j_s) + \sum_{j,j \neq i} \min(E(i_r j_s) - E(i_t j_s)) > 0 \tag{1.4}$$

(1.4) can also be extended to eliminate pairs of rotamers. However, the computation for checking the inequality for pairs takes more time. Goldstein further identified the possible use of super-residues to enhance higher-order elimination. Goldstein also suggested unification [Goldstein, 1994], which merges all rotamers from two positions, that is, constructs a single rotamer out of a pair of rotamers from two positions. This has the effect of exposing pair flags as explicit reduction in the problem size because any single rotamer from a flagged rotamer pair will be discarded in the unified problem. Lasters et al. [1995] developed a linear programming-based approach to exploit the singles elimination condition with respect to a convex combination of rotamers. They also described possible logical deduction between singles and pairs elimination, such as elimination of a single rotamer when every rotamer pair between the single rotamer and all rotamers in another position is flagged. Gordon and Mayo [1998] suggested heuristics for reducing the computational burden of pairs elimination, such as the use of "magic bullet" pair and heuristic scoring of pairs. Pierce et al. [2000] described an algorithm for singles elimination based on the idea of "splitting" the conformational space. They noticed that Goldstein's general elimination condition using a convex combination of single rotamers can be further strengthened by splitting the conformational space, which fixes the rotamer choices for a subset of positions. More recent work by Looger and Hellinga [2001] is called "comparison cluster focusing", seeks to find the smallest residue super-cluster over which all rotamer clusters containing some rotamer c can be eliminated, thus also letting c be eliminated.

Elimination procedures such as Goldstein's conditions and unification [Goldstein, 1994], logical singles-pairs elimination [Lasters et al., 1995], the magic bullet pairs heuristic [Gordon and Mayo, 1998], splitting [Pierce et al., 2000], generalized elimination conditions [Looger and Hellinga, 2001], hybrid optimization through scheduling of various elimination conditions [Gordon et al., 2003], and more recently divide-and-conquer enhancement to DEE [Georgiev et al., 2006] are often able to reduce the problem size dramatically, while demanding only reasonable computational power.

Enhanced DEE [Gordon et al., 2003] that collectively uses various elimination conditions performs well for some of the hard protein design cases of interest to us. However, finding dead-ends using the known elimination conditions does not always eliminate as many rotamers or rotamer pairs as necessary. In case the remaining conformational space after DEE application is too large to literally enumerate, a systematic search method such as the A^* algorithm [Hart et al., 1968, Leach and Lemon, 1998] is often applied to find the GMEC (call the combined method DEE/A^*). However, such a combined scheme will not be useful unless DEE reduces the size of conformational space to the point where a systematic search is applicable.

1.5.2 Other exact methods

Other than DEE, there exist various approaches to solve the GMEC problem exactly. Leach and Lemon [1998], Gordon and Mayo [1999], and Wernisch et al. [2000] describe branch-and-bound (BnB) methods. BnB splits the search space into smaller search spaces and tackles each subproblem at once. Each approach derives a lower bound for the conformational energy, and prunes the ineligible search space by comparing the lower bound with a reference energy (that is, an upper bound).

Eriksson et al. [2001] formulates the side chain positioning problem as an ILP problem. In their computational experiments, they find the LP relaxation of every test instance has an integral solution and, therefore, conclude that an integer programming technique is not necessary. Althaus et al. [2002] presents an integer linear programming (ILP) approach for side-chain demangling in rotamer representation of the side chain conformation. Using an ILP formulation, they identify classes of facet-defining inequalities and devised a separation algorithm for a subclass of inequalities. The results show that the branch-and-cut algorithm is about five times slower than their heuristic approach. Kingsford et al. [2005] also use ILP approaches for side-chain positioning. Yanover et al. [2006] compares the tree-reweighted max-product algorithm and a general purpose LP solver on side-chain prediction and protein design. Sontag and Jaakkola [2007] apply the cutting plane method based on LP to side-chain prediction.

Leaver-Fay et al. [2005] describe a dynamic programming approach based on tree-decomposition. Xu [2005] presents a tree-decomposition algorithm for protein backbone structures. Xie and Sahinidis [2006] describe a method that combines several residue-reduction and rotamer-reduction techniques.

Each exact approach may have some advantages over others depending on the characteristics of the problem being considered. For example, for a simplified version of the problem where the number of rotamers per position is limited or interactions between residue positions are sparse, even deterministic algorithms with guaranteed time bounds exist. However, it is known that protein structures and stabilities can be predicted better with more side-chain flexibility, that is, by using a larger rotamer library [Desjarlais and Handel, 1999, Peterson et al., 2004]. In addition, the network of interactions between residue positions can be dense as is often observed in protein cores. Therefore, we are interested in protein design problems where all possible pairs of positions are assumed to interact and a large number of similar rotamers is offered at each position. To our knowledge, only DEE-like methods or DEE followed by BnB methods have shown success in solving such hard protein design cases exactly.

1.5.3 Approximate methods

There also exist approximate approaches for the GMEC problem. Koehl and Delarue [1994] present the self-consistent mean field theory. Jiang et al. [2000] uses simulated annealing and Monte Carlo sampling to find stable sequences and rotamer combinations. Jones [1994] and Desjarlais and Handel [1995] use genetic algorithms to design proteins for target structures. Wernisch et al. [2000] describe a heuristic for protein design. Ponder and Richards [1987] develop a systematic search method to find side-chain combinations that fit in the cores of small proteins.

Self-consistent mean field theory mentioned above is a probabilistic inference method. The method calculates the mean field energy as the sum of interaction energies weighted by the conformational probabilities. On the other hand, the conformational probabilities are related to the mean field energy by the Boltzmann law. In this method, iterative updates of the probabilities and the mean field energy are performed until both converge numerically. Then, at convergence, the rotamer with the highest probability from each residue is selected as the GMEC. The method is not exact, but takes only linear time per iteration.

Yanover and Weiss [2002] apply belief propagation (BP), generalized belief propagation (GBP), and the mean field method to inferring the GMEC and compared the results with those from SCWRL, a protein-folding program. Using an approximate discrete energy function where only local interactions between neighboring residues are considered, BP and GBP are shown to be more effective in finding the GMEC than other methods considered.

As discussed in Section 1.4, these approximate methods can come up with some solutions faster than the exact methods, but such solutions are not guaranteed to be close to the exact solution. Finding the exact solution is also often required by practitioners for the purpose of improving protein design protocols or comparing solutions from a number of varied design settings.

1.6 Our work and contributions

In this thesis, we present exact solution methods for the GMEC problem. We only consider the search problem when the rotamer choices and associated energy terms are defined. The purpose of our studies is development of exact methods that can be used to solve a large scale GMEC problem on a typical workstation, where the system resources such as CPU power and the size of memory can be limited.

This thesis can be divided into two parts. The first part describes a new BnB method for the GMEC problem. The method consists of several components such as lower bounding through tree-reweighted MAP estimation and branching schemes that make it competitive over the conventional approach using DEE/A^*. The BnB method is evaluated using challenging protein design cases, and is compared with DEE/A^*.

The second part describes Lagrangian dual relaxation approaches to the GMEC problem. In the dual framework, we explore several techniques that can lead to stronger lower bounds than those from the first part based on tree-reweighted MAP estimation. The new lower bounding techniques are again integrated into the BnB method of the first part and evaluated with protein design cases.

1.6.1 Branch-and-bound rotamer optimization using MAP estimation

We describe a new exact solution method for the GMEC problem that can substitute for DEE/A^*, especially in solving hard design cases. Our method, named BroMAP (branch-and-bound rotamer optimization using MAP estimation), is based on the BnB framework and a new subproblem-pruning method. We present lower bounding methods and problem-size reduction techniques, organized into a BnB framework so that BroMAP is guaranteed to find an optimal solution.

Figure 1-6 shows a hierarchical representation of BroMAP. At the highest level, it consists of specifications of bounding and branching schemes. The bound computation of BroMAP mainly relies on a MAP estimation method called the tree-reweighted max-product algorithm (TRMP) [Wainwright et al., 2005]. TRMP can be used to compute both upper and lower bounds, but it is not necessarily more useful for computing upper bounds than other approximate methods mentioned in Section 1.5.3. TRMP is more useful as a lower bounding method in that it is based on the same attainable lower bound as the LP described in Appendix B.3, but is more scalable in computing the lower bound than a general-purpose LP solver. However, such LP bounds are not necessarily strong enough to prune a subproblem in the BnB tree. Therefore, a BnB method simply using TRMP as a lower bounding method is not in general an effective solution method for the GMEC problem.

As suggested by Figure 1-6, the weak lower bounds from TRMP is augmented by what is called "problem-size reduction", or simply reduction. The expected effect of reducing the problem size is that we can obtain a stronger lower bound when applying the same lower bounding technique, such as TRMP, to a smaller problem. A reduced problem induces a smaller search space than the original problem from a reduced number of rotamer choices or positions. However, for the BnB method to work correctly with problem-size reduction, a problem should be reduced only in a way to satisfy a certain condition regarding the relation between the optimal values of the problem before and after the reduction, and the global upper bound. The figure shows that we employ three different reduction techniques that satisfy such a condition. These are DEE modified to function correctly in the BnB framework, rotamer contraction that merges several rotamers as a super-rotamer by identifying rotamers with similar distributions of pairwise energies, and elimination of rotamers and rotamer pairs using individual lower bounds from TRMP.

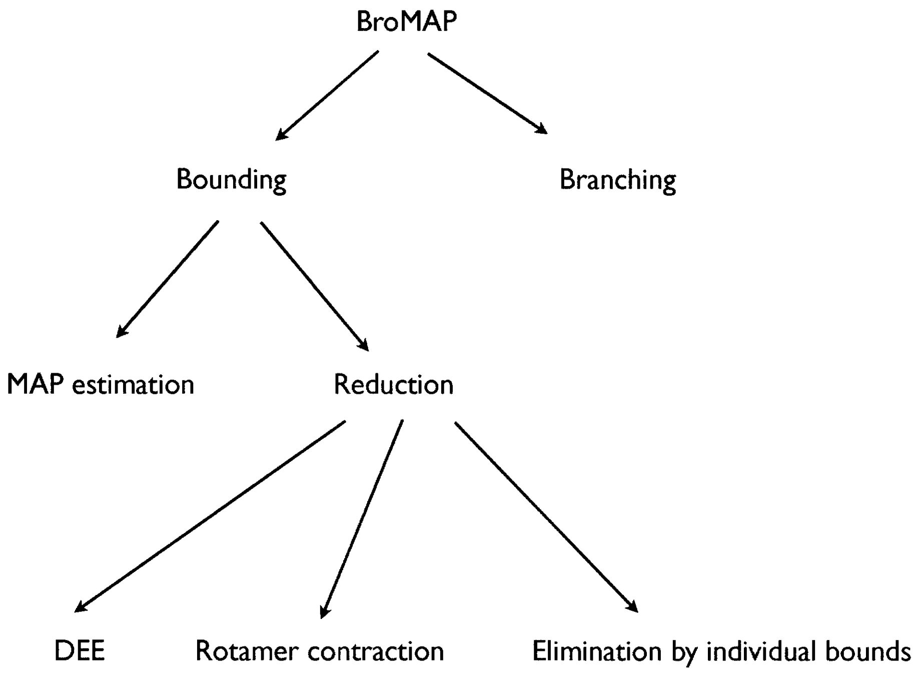


Figure 1-6: A hierarchical representation of BroMAP.

Finally, BroMAP employs a branching scheme for splitting a subproblem and determining the order of nodes expansion in the BnB tree, that exploits rotamer lower bounds from TRMP. The basic idea of BroMAP's branching scheme is separating the optimal values of child subproblems as much as possible and always first expanding those subproblems that are likely to have smaller optimal values. Such a scheme results in fast discovery of an upper bound very close to the global optimal value and effective pruning of subproblems in the BnB tree.

It should be noted that TRMP is a shared component used for lower bounding, elimination by individual bounds, and branching in BroMAP. It enables BroMAP to gain practical performance in solving large scale GMEC problems.

Our numerical experiments confirm the utility of BroMAP in GMEC search for large protein design problems, including ones that are challenging for DEE/A^*. In our experiments, all cases solved by DEE/A^* were also solved by BroMAP, and using BroMAP did not incur significant disadvantage over DEE/A^*. Moreover, BroMAP excelled on the cases where DEE/A^* did not perform well; for each case that took longer than one hour but was eventually solved by DEE/A^*, BroMAP took at most 33% of the DEE/A^* running time. Among 68 test cases of various types and sizes, we found BroMAP failed to solve three cases within the 7-day allowed time whereas DEE/A^* failed to solve 17 of them.

Compared to DEE, BroMAP has an advantage that it can attack smaller subproblems separately using various problem-size reduction or lower bounding techniques instead of having to keep the problem as a whole. Meanwhile, the use of DEE as one of the problem-size reduction techniques in BroMAP allows the strengths of DEE for protein design problems to be transferred to BroMAP.

BroMAP has the advantage of reducing the search trees over conventional BnB approaches in two ways. First, it uses problem-size reduction techniques within each node so that the effect of problem-size reduction from branching is often larger than that of a conventional BnB method. Hence, the depth of the resulting search tree is also smaller. Second, it quickly finds a strong upper bound (at the end of the first depth-first dive) with the help of informed branching and subproblem selection. This facilitates effective pruning of nodes that follow, and therefore often results in sparse search trees growing mostly in one direction. BroMAP achieves these advantages without excessive computation by using new inexpensive lower bounding methods and limiting the effort spent by bounding or problem-size reduction.

The following are the contributions made in the first part of this thesis:

1. Development of lower bounding methods for minimum conformation energy of individual rotamers and rotamer pairs using a maximum-a-posteriori estimation method called the tree-reweighted max-product algorithm [Wainwright et al., 2005];

2. Adoption of problem-size reduction techniques (DEE and elimination by lower bounds) within the BnB framework;

3. Use of rotamer lower bounds in branching and subproblem selection for fast discovery of strong upper bounds;

4. Extensive evaluation of BroMAP and DEE/A^* on various types and sizes of protein design problems.

1.6.2 Lagrangian dual relaxation of the GMEC problem

The results from computational experiments in the first part suggest a speed-up of the BnB method can be obtained through stronger lower bounding techniques. Motivated by this observation, we explore Lagrangian dual relaxation approaches to the GMEC problem. Using the dual optimization framework provides several advantages, but its scalability is the biggest gain over its primal counterpart, such as linear programming approaches.

We suggest two different approaches to strengthening lower bounds in the dual framework. The first approach uses inclusion of higher order cliques, such as triplets. The effect of including relevant triplets is experimentally shown using protein design cases.

The second approach to improve lower bounds in the dual framework is sought from identification and addition of violated valid cuts. We suggest a new iterative lower bound tightening approach for the GMEC problem through addition of cuts to the Lagrangian dual problem, which is a dual analog of the cutting plane method in the primal domain. Lower bounding the GMEC energy via the iterative method is probed with respect to the quality of bounds and the bounding time.

We finally describe a method of computing individual rotamer and rotamer-pair lower bounds in the dual framework. Coupled with the strong bounding ability from addition of triplets, the resulting individual bounds are sometimes stronger than individual bounds from TRMP, but are not always stronger due to the suboptimal convergence of the subgradient method.

The dual method including individual bounds computation is employed in BroMAP and evaluated on small test cases. The computational results suggest that the dual method needs speed improvement, despite its stronger bounding ability, for BroMAP employing the dual bounds to be competitive.

The following are the contributions made in the second part:

1. Formal description of the Lagrangian dual relaxation and its solution method when cyclic subgraphs are used for decomposition of the original GMEC problem.

2. A heuristic method for finding relevant triplets to improve the lower bound in the dual framework.

3. Development of a new iterative method for tightening the lower bound of the GMEC problem through addition of violated valid cuts in the dual framework.

4. Development of lower bounding methods for individual rotamers and rotamer-pairs in the dual framework.

5. Evaluation of the Lagrangian dual lower bounding methods in the BnB framework to solve protein design cases.

1.7 Overview

In Chapter 2, we cast the GMEC problem as the MAP estimation problem, and review solution methods of the MAP estimation problem. In Chapter 3, we describe a BnB method (BroMAP) for the GMEC problem, and its components such as lower bounding techniques, elimination methods, and branching schemes. Chapter 4 suggests problem-size reduction and lower bounding techniques that can be used under the BroMAP framework. In Chapter 5, we apply BroMAP to challenging protein design cases, and compare its performance to that of DEE/A^*. Chapter 6 describes Lagrangian dual relaxation of the GMEC problem, and a subgradient method as a solution method for the dual problem. The effect of including triplets in the dual formulation on the strength of lower bounds is evaluated. Chapter 7 presents an iterative lower bound tightening method based on identification of violated valid cuts in the dual framework. The suggested method is evaluated on protein design cases. Chapter 8 describes a method to compute lower bounds of individual rotamers and rotamer-pairs. The individual lower bounding methods built on the strengthened dual lower bounding methods of the previous chapters are adopted in the BroMAP framework, replacing the lower bounding through tree-reweighted MAP estimation. The resulting BnB method is evaluated with protein design cases.

Chapter 2

Background on MAP estimation problem and methods

Probabilistic inference approaches have been popularly applied to a wide range of applications such as computer vision, computational biology, and medical decision making, to name a few. A multinomial inference problem involves a random vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ characterized by a probability distribution that maps a sample $\mathbf{x}$ in a sample space $\mathcal{X}$ to a probability $p(\mathbf{x})$ [Cowell et al., 1999]. There are various types of inference problems such as computing the marginal distribution $p(x_A) = \sum_{\mathbf{x} \setminus x_A} p(\mathbf{x})$ for some $x_A \subset \mathbf{x}$. In this chapter, we formulate the GMEC problem as another type of probabilistic inference problem called *maximum-a-posteriori* (MAP) estimation problem. The formulation will be presented with the framework of graphical models. Then, we review known MAP estimation methods including the conventional max-product algorithm, and the tree-reweighted max-product (TRMP) algorithm. Particularly, we expand on TRMP, which will be the main tool for obtaining upper and lower bounds of the GMEC problem in Chapter 3. The main idea of TRMP is to express a distribution defined over a cyclic graphical model as a convex combination of distributions over spanning trees. This convex combination of tree distributions is used to upper bound the MAP probability, that is, to lower bound the GMEC energy. Finding an assignment that maximizes such an upper bound is computationally much easier than computing the exact MAP assignment of the original distribution. After reviewing TRMP, we briefly introduce other known MAP estimation methods. Particularly, we review mean field theory and the sum-product algorithm that are often heuristically used to compute an approximate MAP assignment, and point out connections to other MAP estimation methods.

2.1 MAP estimation for the GMEC problem

The MAP estimation problem asks to find a MAP assignment $\mathbf{x}^*$ such that

$$\mathbf{x}^* \in \arg\max_{\mathbf{x}\in\mathcal{X}} p(\mathbf{x}), \tag{2.1}$$

where $\mathcal{X}$ is the sample space for $\mathbf{x}$. In the GMEC problem, we number the sequence positions by $i = 1, \ldots, n$, and associate with each position i a discrete random variable x_i that ranges over R_i, a set of allowed rotamers at position i. Then, we can define a probability distribution $p(\mathbf{x})$ over

$$\mathcal{X} = R_1 \times \ldots \times R_n \tag{2.2}$$

as

$$p(\mathbf{x}) = \frac{1}{Z}\exp\{-e(\mathbf{x})\}, \tag{2.3}$$

for a normalization constant Z and

$$e(\mathbf{x}) = \sum_{i=1}^{n} e_i(x_i) + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} e_{ij}(x_i, x_j), \tag{2.4}$$

where

$$\begin{aligned} e_i(r) &= E(i_r) \text{ for } r \in R_i, && (2.5)\\ e_{ij}(r,s) &= E(i_r j_s) \text{ for } (r,s) \in R_i \times R_j. && (2.6) \end{aligned}$$

Therefore, the GMEC problem for minimizing $e(\mathbf{x})$ is equivalent to the MAP estimation problem for $p(\mathbf{x})$.

2.2 A graphical model of the GMEC problem

A probability distribution over a random vector can be related to a graphical model [Cowell et al., 1999] (In turn, a graphical model represents a family of many probability distributions.) An undirected graphical model $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a set of vertices $\mathcal{V}$ that represent random variables and a set of edges $\mathcal{E}$ connecting some pairs of vertices. The connectivity between vertices, i.e. $\mathcal{E}$, is determined by the conditional independencies between random variables of the corresponding distribution. That is, a probability distribution $p(\mathbf{x})$ can be represented by an undirected graphical model $\mathcal{G}$ if $p(\mathbf{x})$ can be factorized into non-negative functions, called compatibility functions, each of which is defined over variables in a clique of $\mathcal{G}$. More precisely, if we define compatibility functions $\psi_C(x_C) : \prod_{v_i \in \mathcal{V}(C)} R_{v_i} \mapsto \mathbb{R}_+$ for each maximal clique C in $\mathcal{G}$ (i.e. $\mathcal{V}(C) \subset \mathcal{V}$ and $\mathcal{E}(C) \subset \mathcal{E}$), then the distribution can be factorized as

$$p(\mathbf{x}) = \frac{1}{Z} \prod_C \psi_C(x_C), \tag{2.7}$$

for a normalization constant Z.

Graphical models serve not only as a means of visualizing the conditional independence, but also enable inference problems to be approached from the perspective of graph theory. In finding a general solution to inference problems, the structure of the graphical models often becomes a useful measure of complexity and provides a tool for systematic exploration of the distribution. Therefore, when we refer to an inference problem, describing the structure of the associated graphical model conveys a good amount of information on the problem's complexity. Therefore, we will often denote distributions by their associated graphical model; for example, a "tree distribution" refers to a distribution represented by a tree graphical model.

In our MAP estimation equivalent of the GMEC problem, given the potential energy in the form of (2.4), the probability distribution (2.3) can be rewritten as

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^{n} e^{-E_i(x_i)} \prod_{i=1}^{n-1} \prod_{j=i+1}^{n} e^{-E_{ij}(x_i, x_j)}. \tag{2.8}$$

The presence of factors joining each pair of variables indicates that the graphical model that corresponds to (2.8) is complete undirected graph with $|\mathcal{V}| = n$ vertices. The typical motivation for using the graphical model is finding as simple a model as possible that captures conditional independencies among variables. However, we generally consider a complete graph with n vertices as the graphical model for the GMEC problem, that is, the protein design problems we are interested in have molecular interactions between every pair of positions. For these cases the factorization or graphical representation does not directly help in solving the inference problems.

2.3 Max-marginals and max-product algorithms

In this section, we first introduce the notion of max-marginal, which become useful terms when describing the operation of MAP estimation methods. Then, we introduce a conventional MAP estimation method called max-product algorithm [Cowell et al., 1999], and briefly introduce a variant called tree reparamterization max-product algorithm [Wainwright et al., 2004]. These algorithms are useful in upper bounding the GMEC energy $e(\mathbf{x}^*)$ (or lower bounding the MAP probability $p(\mathbf{x}^*)$).

2.3.1 Max-marginals

Wainwright et al. [2004] define (singleton) max-marginals μ_i as the maximum of $p(\mathbf{x})$ when one of the variables x_i is constrained to a specific value, multiplied by some constant $\kappa_i > 0$, i.e.

$$\mu_i(x_i) = \kappa_i \max_{\{\mathbf{x}' | x'_i = x_i\}} p(\mathbf{x}'). \tag{2.9}$$

Similarly, pairwise max-marginals μ_{ij} are defined as

$$\mu_{ij}(x_i, x_j) = \kappa_{ij} \max_{\{\mathbf{x}' | x'_i = x_i, x'_j = x_j\}} p(\mathbf{x}'), \tag{2.10}$$

the maximum of $p(\mathbf{x})$ when a pair of the variables are constrained to a specific pair of values. Note that κ_i and κ_{ij} are constants that can vary depending on i and j. In what follows, we will simply denote all the constants as κ. It is known that any tree

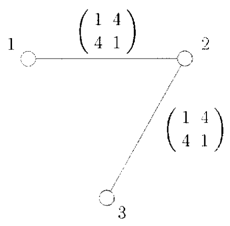

1
2
3
$\begin{pmatrix}1&4\\4&1\end{pmatrix}$
$\begin{pmatrix}1&4\\4&1\end{pmatrix}$

Figure 2-1: The diagram shows the graphical model and pairwise compatibility functions $\psi_{12}(x_1, x_2)$ and $\psi_{23}(x_2, x_3)$ of the distribution used in Example 3.

distribution $p(\mathbf{x})$ can be factorized in terms of its max-marginals as [Cowell et al., 1999]

$$p(\mathbf{x}) \propto \prod_{i \in \mathcal{V}} \mu_i(x_i) \prod_{(i,j)\in\mathcal{E}} \frac{\mu_{ij}(x_i, x_j)}{\mu_i(x_i)\mu_j(x_j)}. \tag{2.11}$$

If we knew the max-marginals of a tree distribution $p(\mathbf{x})$, we could easily compute the maximum value of $p(\mathbf{x})$.

Example 1. *(Max-marginals) [Wainwright et al., 2004]. Let* $\mathbf{x} \in \{0, 1\}^3$ *be a random vector defined by a graphical model of Figure 2-1 and compatibility functions* ψ *such that*

$$\psi_i(x_i) = 1, \textit{ for all } x_i \in \{0,1\} \textit{ and } i \in \{1,2,3\}, \tag{2.12}$$

and

$$\psi_{ij}(x_i, x_j) = \begin{cases} 1 & \textit{if } x_i = x_j \\ 4 & \textit{otherwise} \end{cases} \quad \textit{for all } (i,j) \in \{(1,2),(2,3)\}. \tag{2.13}$$

That is, $p(\mathbf{x}) = \frac{1}{50}\psi_1(x_1)\psi_2(x_2)\psi_3(x_3)\psi_{12}(x_1,x_2)\psi_{23}(x_2,x_3)$

Then, it is easy to verify $\max_{\{\mathbf{x}'|x_1'=x_1\}} p(\mathbf{x}') = 4^2/50$ *for all* $x_1 \in \{0,1\}$. *Therefore, we can define max-marginals* $\mu_1(x_1) = 1$ *for all* $x_1 \in \{0,1\}$, *i.e.* $\max_{\{\mathbf{x}'|x_1'=x_1\}} p(\mathbf{x}') = \frac{4^2}{50}\mu_1(x_1)$ *and* $\kappa_1 = \frac{50}{4^2}$. *Since* $\mu_2(x_2)$ *and* $\mu_3(x_3)$ *can be defined similarly, we obtain* $\mu_i(x_i) = 1$ *for all* $x_i \in \{0,1\}$ *and* $i \in \{1,2,3\}$.

Likewise, we can verify $\max_{\{\mathbf{x}'|(x_1',x_2')=(x_1,x_2)\}} p(\mathbf{x}')$ *is* $4/50$ *if* $x_1 = x_2$, *and* $4^2/50$ *otherwise. Since we obtain the same result when maximizing under fixed* (x_2, x_3)

values, we can define $\mu_{ij}(x_i, x_j)$ as

$$\mu_{ij}(x_i, x_j) = \begin{cases} 1 & \textit{if } x_i = x_j \\ 4 & \textit{otherwise} \end{cases} \quad \textit{for all } (i,j) \in (1,2),(2,3). \tag{2.14}$$

i.e. $\max_{\{\mathbf{x}'|(x_i',x_j')=(x_i,x_j)\}} p(\mathbf{x}') = \frac{4}{50}\mu_{ij}(x_i,x_j)$ *and* $\kappa_{ij} = \frac{50}{4}$.

In this example, we realize $\mu_i(x_i) = \psi_i(x_i)$ and $\mu_{ij}(x_i, x_j) = \psi_{ij}(x_i, x_j)$ for all i, j, and also $\mu_{ij}(x_i, x_j) = \psi_{ij}(x_i, x_j)\psi_i(x_i)\psi_j(x_j)$. This makes us easily verify that $p(\mathbf{x})$ is factorized by max-marginals:

$$\begin{aligned} p(\mathbf{x}) &= \frac{1}{50}\psi_1(x_1)\psi_2(x_2)\psi_3(x_3)\frac{\psi_{12}(x_1,x_2)\psi_1(x_1)\psi_2(x_2)}{\psi_1(x_1)\psi_2(x_2)}\frac{\psi_{23}(x_2,x_3)\psi_2(x_2)\psi_3(x_3)}{\psi_2(x_2)\psi_3(x_3)} && (2.15) \\ &= \frac{1}{50}\mu_1(x_1)\mu_2(x_2)\mu_3(x_3)\frac{\mu_{12}(x_1,x_2)}{\mu_1(x_1)\mu_2(x_2)}\frac{\mu_{23}(x_2,x_3)}{\mu_2(x_2)\mu_3(x_3)}. && (2.16) \end{aligned}$$

Now, assume that we are given $p(\mathbf{x})$ and the max-marginals $\{\mu_i, \mu_{ij}\}$. We illustrate how max-marginals can be used to compute $\max_{\mathbf{x}} p(\mathbf{x})$. We know $p(\mathbf{x}) = \frac{1}{Y}\mu_1(x_1)\mu_2(x_2)\mu_3(x_3)\frac{\mu_{12}(x_1,x_2)}{\mu_1(x_1)\mu_2(x_2)}\frac{\mu_{23}(x_2,x_3)}{\mu_2(x_2)\mu_3(x_3)}$ for some Y. The value of Y can be easily computed by comparing both sides of the equation for some specific assignment, e.g. $(0,0,0)$. In this example, we obtain $Y = 50$ as shown in (2.16). Assuming $\mathbf{x}^*$ is a MAP assignment, we have

$$\max_{\mathbf{x}} p(\mathbf{x}) = p(\mathbf{x}^*) = \frac{1}{50}\mu_1(x_1^*)\mu_2(x_2^*)\mu_3(x_3^*)\frac{\mu_{12}(x_1^*,x_2^*)}{\mu_1(x_1^*)\mu_2(x_2^*)}\frac{\mu_{23}(x_2^*,x_3^*)}{\mu_2(x_2^*)\mu_3(x_3^*)}. \tag{2.17}$$

Since we know x_i^* and (x_i^*, x_j^*) should be a maximizer of $\mu_i(x_i)$ and $\mu_{ij}(x_i, x_j)$, respectively, the maximum value of $p(\mathbf{x})$ can be obtained simply by finding the maximum value of each $\mu_i(x_i)$ and $\mu_{ij}(x_i, x_j)$ without needing to find the actual assignment $\mathbf{x}^*$. Therefore, $\max_{\mathbf{x}} p(\mathbf{x}) = 4^2/50$.

□

2.3.2 Conventional max-product algorithm

The max-product algorithm (also known as max-plus or min-sum) is an iterative algorithm that estimates MAP configuration by propagating a series of "messages" between pairs of vertices in the graphical model [Cowell et al., 1999, Wainwright et al., 2004]. Vertex $i \in \mathcal{V}$ sends to each of its neighbor vertex $j \in \mathcal{V}$, $(i, j) \in \mathcal{E}$, the message $m_{ij}(x_j)$ which can be interpreted as how likely vertex i sees node j is in state x_j. For tree distributions, it can be shown that values of messages between every pair of nodes converge after a finite number of iterations [Pearl, 1988]. The converged messages $m^* = \{m^*_{ij}\}$ then define functions $\{q_i\}$ for each $i \in \mathcal{V}$:

$$q^*_i(x_i) = \kappa \psi_i(x_i) \prod_{j \in N(i)} m^*_{ji}(x_i), \tag{2.18}$$

where κ is some constant that depends on i. It is known that, in tree distributions, these functions are in fact equivalent to singleton max-marginals $\{\mu_i\}$ [Cowell et al., 1999, Wainwright et al., 2004]. Given this fact, it is obvious that we can find the MAP assignment $\mathbf{x}^*$ by assigning the maximizer of $q^*_i(x_i)$ as x^*_i when each $q^*_i(x_i)$ has a unique maximizer:

$$x^*_i = \arg\max_{x_i} q^*_i(x_i). \tag{2.19}$$

On the other hand, functions q_{ij} analogous to (2.18) but with pairwise variables are defined by

$$q^*_{ij}(x_i, x_j) = \kappa \psi_i(x_i) \psi_j(x_j) \psi_{ij}(x_i, x_j) \prod_{l \in N(i) \setminus j} m^*_{li}(x_i) \prod_{l \in N(j) \setminus i} m^*_{lj}(x_j). \tag{2.20}$$

When the max-product is applied to tree distributions, it is also known that the q^*_{ij} is equivalent to pairwise max-marginals μ_{ij}.

When the singleton max-marginals have multiple maximizers, assignment by (2.19) is not generally a correct MAP assignment. Max-marginals are still useful for finding a MAP assignment when the distribution is tree. This can be done in the following way, starting from a designated root node of the tree:

1. if the current node is i is the root, randomly pick a maximizer of the root node's singleton max-marginals, and designate it as the assignment.

2. otherwise, pick a maximizer x_i^* of the current node's singleton max-marginals such that the pairwise assignment (x_i^*, x_j^*) also maximizes the pairwise max-marginals of $(i, j) \in \mathcal{E}$, where j is the parent of i.

3. Repeat step 2 for all child nodes of i.

Note that the above procedure is not applicable even with correct max-marginals when the graphical model contains cycles [Wainwright et al., 2004]

Summarizing the results on max-marginals computation by the max-product algorithm above, it is obvious that the max-product algorithm can find the exact MAP assignment for tree distributions in a finite number of iterations. On the other hand, since the max-product algorithm is simply described in terms of message passing and update routines between pair of vertices, it can be also freely applied to cyclic graphical models, such as the one for the GMEC problem. In this case, however, the algorithm does not generally find the MAP assignment. In addition, convergence of the algorithm is not predictable either. However, there are various empirical results that report the effectiveness of the max-product algorithm in finding the exact MAP configuration or at least a good estimation of it for cyclic graphical models [Freeman and Pasztor, 2000, Benedetto et al., 1996, Weiss, 1997].

When the max-product algorithm converges on a graphical model with cycles, we can construct a max-product assignment $\mathbf{x}_{MP}$ by (2.19), which is not guaranteed to be a MAP assignment, $\mathbf{x}^*$. It is known that $\mathbf{x}_{MP}$ is a MAP assignment when the graph contains at most one cycle, and also that $\mathbf{x}_{MP}$ is at least a local optimum for an arbitrary topology of graphical model [Freeman and Weiss, 2000]. Wainwright et al. [2004] extended this result to show that $\mathbf{x}_{MP}$ is optimal for every subgraph with at most one cycle under a particular condition.

2.3.3 Tree reparameterization max-product

Tree reparameterization max-product algorithm [Wainwright et al., 2004] is another form of the max-product algorithm constructed from the view of reparameterization. The term "reparameterization" comes from the fact that a distribution $p(\mathbf{x})$ over a graphical model can be represented by different sets of factors. For example, a distribution defined by a graphical model and compatibility functions ψ as in (2.7) can be represented as a product of differently defined compatibility functions. Tree reparameterization max-product attempts to solve the MAP estimation problem by repeatedly reparameterizing the distribution with respect to each spanning tree at each iteration. The results of reparameterization at each iteration is factors corresponding to max-marginals with respect to the tree being considered at the iteration, and residual factors corresponding to the rest of the graphical model.

Similarly to the conventional max-product algorithm, tree reparamterization max-product does not generally find the exact MAP assignment of a cyclic graphical model. However, our computational experiences suggest that the latter has a better convergence behavior, and the assignments it finds are often better than those from the conventional max-product algorithm in terms of conformational energy.

2.4 Pseudo-max-marginals and tree-reweighted max-product algorithm

Despite the empirically proven performance of the max-product algorithms in practice, lack of knowledge on the exactness of MAP estimation makes the algorithms less attractive. The recent work by Wainwright et al. [2005] presents a new algorithm called tree-reweighted max-product algorithm (TRMP). The most appealing feature of the algorithm is that it may occasionally confirm the optimality of assignment it finds at convergence. We start the discussion of TRMP by introducing the notion of pseudo-max-marginal, which will be frequently used in describing TRMP. The pseudo-max-marginal can be regarded as an extended counterpart of max-marginal of the max-product algorithms.

2.4.1 Pseudo-max-marginals

For general non-tree (cyclic) distributions, there is no known method that efficiently computes max-marginals; computing them can be as expensive as the original MAP estimation problem. In addition, for distributions over non-tree graphical models, knowing exact max-marginals is not generally useful either to obtain a MAP assignment or to compute the maximum value of $p(\mathbf{x})$[1]. Instead, Wainwright et al. [2005] use the notion of pseudo-max-marginals in their tree-reweighted max-product (message-passing) algorithm.

The basic idea of the tree-reweighted max-product algorithm is to express the non-tree distribution as a convex combination of distributions over a set of spanning trees. This convex combination of tree distributions is used to upper bound the MAP probability, that is, to lower bound the energy. Finding an assignment that maximizes the upper bound is computationally much easier than computing the exact MAP assignment of the original distribution. It can be shown that the upper bound is tight if and only if every tree distribution shares a common MAP configuration, i.e. tree agreement [Wainwright et al., 2005]. In that case, the shared configuration is also the MAP configuration of the original distribution. The tree-reweighted max-product algorithm tries to induce this tree agreement by factorizing each tree distribution with factors called pseudo-max-marginals and having pseudo-max-marginals converge to the max-marginals of each tree distribution.

Let us assume we use the tree-reweighted max-product algorithm with $\mathcal{T}$, a set of spanning trees of $\mathcal{G}$, and some non-negative constant $\rho(T)$ for each $T \in \mathcal{T}$ such that $\sum_{T \in \mathcal{T}} \rho(T) = 1$. The tree-reweighted max-product algorithm requires that every vertex and edge of $\mathcal{G}$ be covered by $\mathcal{T}$, i.e. each vertex and edge in $\mathcal{G}$ is in some tree T in $\mathcal{T}$ such that $\rho(T) > 0$. Then, by construction, pseudo-max-marginals $\nu = \{\nu_i, \nu_{ij}\}$ from the tree-reweighted max-product algorithm satisfy "ρ-reparameterization", that

[1]Exact max-marginals are useful for finding a MAP assignment or computing the MAP probability in the absence of tied values for the same variable

is described as:

$$p(\mathbf{x}) \propto \prod_{T \in \mathcal{T}} \left[\prod_{i \in \mathcal{V}(T)} \nu_i(x_i) \prod_{(i,j) \in \mathcal{E}(T)} \frac{\nu_{ij}(x_i, x_j)}{\nu_i(x_i)\nu_j(x_j)} \right]^{\rho(T)} = \prod_{i \in \mathcal{V}} \nu_i(x_i)^{\rho_i} \prod_{(i,j) \in \mathcal{E}} \left[\frac{\nu_{ij}(x_i, x_j)}{\nu_i(x_i)\nu_j(x_j)} \right]^{\rho_{ij}}, \tag{2.21}$$

where ρ_{ij} is an edge coefficient such that $\rho_{ij} = \sum_{T \in \mathcal{T}:(i,j) \in \mathcal{E}(T)} \rho(T)$ for all $(i,j) \in \mathcal{E}$, and ρ_i is a vertex coefficient such that $\rho_i = \sum_{T \in \mathcal{T}: i \in \mathcal{V}(T)} \rho(T)$ for all $i \in \mathcal{V}$. Note that, if $\mathcal{T}$ is a set of spanning trees, then ρ_i is 1 for all $i \in \mathcal{V}$.

A tree distribution $p^T(\mathbf{x}; \nu)$ for some $T \in \mathcal{T}$ and given pseudo-max-marginals can be defined as

$$p^T(\mathbf{x}; \nu) = \prod_{i \in \mathcal{V}(T)} \nu_i(x_i) \prod_{(i,j) \in \mathcal{E}(T)} \frac{\nu_{ij}(x_i, x_j)}{\nu_i(x_i)\nu_j(x_j)}. \tag{2.22}$$

Then, we have

$$p(\mathbf{x}) \propto \prod_{T \in \mathcal{T}} \{p^T(\mathbf{x}; \nu)\}^{\rho(T)} \tag{2.23}$$

from (2.21). The pseudo-max-marginals ν^* at convergence of the tree-reweighted max-product algorithm satisfy the "tree-consistency condition" with respect to every tree $T \in \mathcal{T}$. That is, the pseudo-max-marginals converge to the max-marginals of each tree distribution.

Example 2. *(Pseudo-max-marginals) [Wainwright et al., 2005]. Let* $\mathbf{x} \in \{0,1\}^3$ *be a random vector on a graphical model illustrated in Figure 2-2(a). Let*

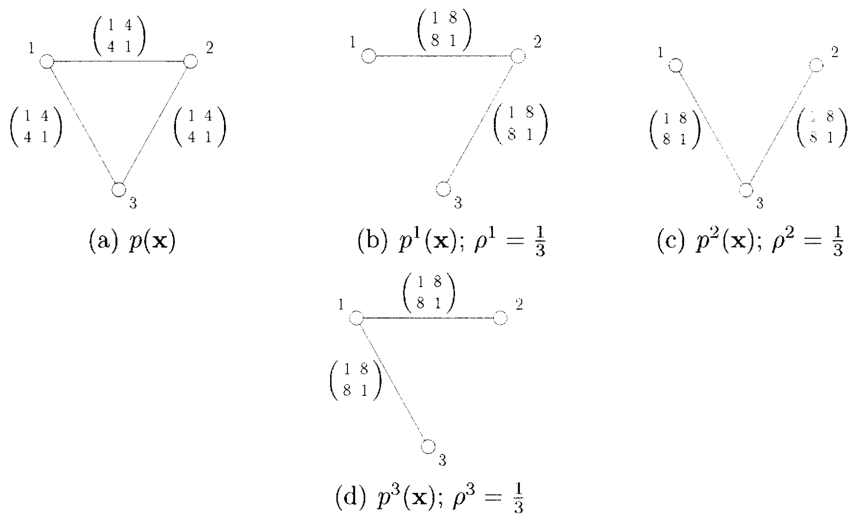
$$p(\mathbf{x}) = \frac{1}{98} \psi_1(x_1)\psi_2(x_2)\psi_3(x_3)\psi_{12}(x_1, x_2)\psi_{23}(x_2, x_3)\psi_{31}(x_3, x_1), \tag{2.24}$$

where $\psi_i(x_i)$ *and* $\psi_{ij}(x_i, x_j)$ *are defined same as in Example 1. We define pseudo-max-marginals* $\hat{\nu}$ *as follows:*

$$\hat{\nu}_i(x_i) = 1, \textit{ for all } x_i \in \{0,1\} \textit{ and } i \in \{1,2,3\}, \tag{2.25}$$

$$\hat{\nu}_{ij}(x_i, x_j) = \begin{cases} 1 & \textit{if } x_i = x_j \\ 8 & \textit{otherwise} \end{cases} \quad \textit{for all } (i,j) \in \{(1,2),(2,3),(3,1)\}. \tag{2.26}$$

Figure 2-2(b) – (d) illustrates the trees used for the convex combination and



$\begin{pmatrix}1&4\\4&1\end{pmatrix}$
$\begin{pmatrix}1&8\\8&1\end{pmatrix}$
1
2
3
(a) $p(\mathbf{x})$
(b) $p^1(\mathbf{x})$; $\rho^1 = \frac{1}{3}$
(c) $p^2(\mathbf{x})$; $\rho^2 = \frac{1}{3}$
(d) $p^3(\mathbf{x})$; $\rho^3 = \frac{1}{3}$

Figure 2-2: Illustration of pseudo-max-marginals and ρ-reparameterization. (a) Original distribution. (b) – (d) Pseudo-max-marginals on each tree used by convex combination.

pseudo-max-marginals on each tree. It can be easily verified that pseudo-max-marginals on each tree are in fact max-marginals. Thus, the pseudo-max-marginals are tree-consistent. The distribution for each tree is given by (2.22). For example, the distribution for Figure 2-2(b) is

$$p^1(\mathbf{x}) = \hat{\nu}_1(x_1)\hat{\nu}_2(x_2)\hat{\nu}_3(x_3)\frac{\hat{\nu}_{12}(x_1,x_2)}{\hat{\nu}_1(x_1)\hat{\nu}_2(x_2)}\frac{\hat{\nu}_{23}(x_2,x_3)}{\hat{\nu}_2(x_2)\hat{\nu}_3(x_3)}. \tag{2.27}$$

Then, by letting $\rho(T) = 1/3$ for all three trees, we obtain

$$\frac{1}{98}p^1(\mathbf{x})^{1/3}p^2(\mathbf{x})^{1/3}p^3(\mathbf{x})^{1/3} \tag{2.28}$$

$$= \frac{1}{98}\psi_1(x_1)\psi_2(x_2)\psi_3(x_3)\psi_{12}(x_1,x_2)\psi_{23}(x_2,x_3)\psi_{31}(x_3,x_1) = p(\mathbf{x}), \tag{2.29}$$

from $\psi_i(x_i) = \hat{\nu}_i(x_i)^{-1/3}$ and $\psi_{ij}(x_i,x_j) = \hat{\nu}_{ij}(x_i,x_j)^{2/3}$. This verifies the pseudo-max-marginals satisfy ρ-reparameterization as well.

□

2.4.2 Tree-reweighted max-product

Wainwright et al. [2005] present two methods for maximizing the convex combination of tree distributions: linear programming (LP) and message-passing. Though both are based on common analytical results, the appearance of the algorithms and their applicabilities are different. Simply solving LP will output the MAP probability when the upper bound is tight, and also a MAP assignment if the optimality specification criteria are met. However, the message-passing algorithm may or may not find the MAP configuration even for the same problem due to numerical issues with finding the fixed point. However, the message-passing algorithm scales better than LP and often converges to sub-optimal configurations even when the upper bound is not tight (so LP fails).

Algorithm 12 in Appendix A describes "edge-based reparameterization updates" [Wainwright et al., 2005] defining $\mathcal{T}$ as a set of (not necessarily spanning) trees in $\mathcal{G}$, as used by Kolmogorov [2006]. In what follows, we will call this algorithm TRMP in short. Note that, although we define $\mathcal{T}$ as a set of general trees covering all vertices and edges of $\mathcal{G}$, it can be easily verified that all the analyses done by Wainwright et al. [2005] can be applied to TRMP in exactly the same way, to show TRMP has the same properties owned by the original edge-based reparameterization updates.

TRMP can sometimes guarantee the optimality of an assignment found at convergence for non-tree distributions. Even if TRMP does not find the exact MAP assignment, we can easily compute the exact maximum value for each tree distribution at TRMP convergence since pseudo-max-marginals converge to max-marginals for each tree distribution. Then, we can combine these to get an upper bound for the original, non-tree distribution (thereby obtaining a lower bound on the energy).

We are free to choose any set of trees $\mathcal{T}$ and $\rho(\cdot)$ as long as each vertex and edge is covered by some $T \in \mathcal{T}$ with $\rho(T) > 0$. In this work, we consistently use a set of maximal stars $\mathcal{S}$ in place of $\mathcal{T}$ for the convenience of implementation and the simplicity in computing rotamer/rotamer-pair lower bounds. A star is a tree where at most one vertex is not a leaf. We denote the center of star S as $\gamma(S)$. A maximal

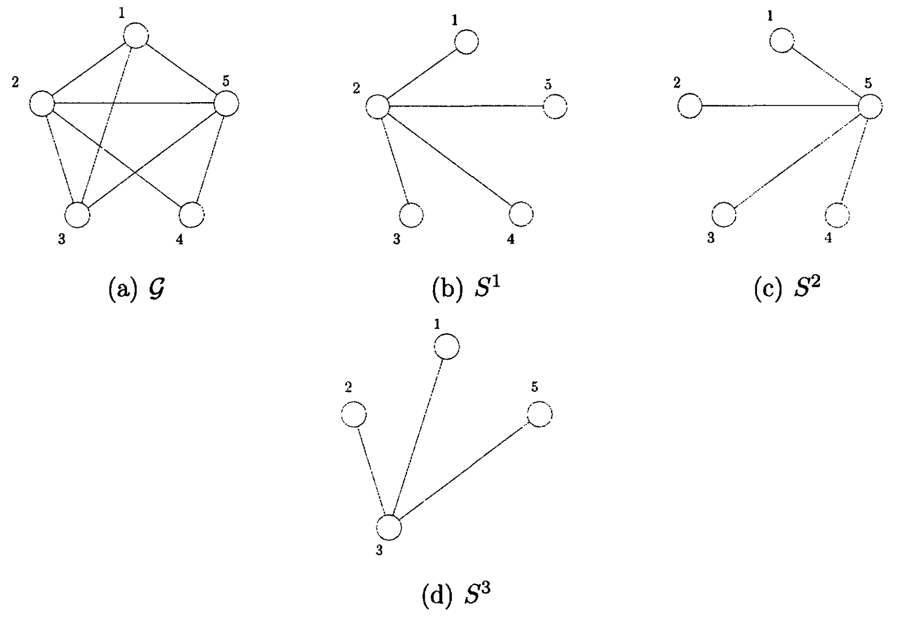


Figure 2-3: Example of covering a graph by maximal stars: $\mathcal{G}$ of (a) is completely covered by S^1, S^2, and S^3.

star is a star that is not a subset of another star. Figure 2-3 illustrates covering a graph by a set of maximal stars; all vertices and edges of graph (a) are covered by $\mathcal{S}$ consisting of three maximal stars shown in (b), (c), and (d).

Following the terminology of Kolmogorov [2006], we say ν is in a normal form if it satisfies $\max_{r \in R_i} \nu_i(r) = 1$ for all $i \in \mathcal{V}$, and $\max_{(r,s) \in R_i \times R_j} \nu_{ij}(r,s) = 1$ for all $(i,j) \in \mathcal{E}$. In what follows, we assume ν^n of Algorithm 12 is always in a normal form. Then, using (2.21) and some constant ν_c^n that may vary with iteration n of TRMP, we obtain

$$p(\mathbf{x}) = \nu_c^n \prod_{S \in \mathcal{S}} \{p^S(\mathbf{x}; \nu^n)\}^{\rho(S)}. \tag{2.30}$$

The value of ν_c^n can be easily computed by comparing both sides of (2.30) for any assignment $\mathbf{x} \in \mathcal{X}$ and ν^n.

2.5 Other methods for MAP estimation

Since the work of Wainwright et al. [2005] on MAP estimation on the undirected graphical model, various contributions have been made in developing new MAP estimation methods that guarantee finding the optimal solution under certain conditions. Kolmogorov [2006] describes the sequential tree-reweighted message passing algorithm, which guarantees the convergence to the so-called "weak tree agreement", and monotonic improvement of the bound. Globerson and Jaakkola [2007a] developed a message passing algorithm that solves the dual of the LP relaxation of the MAP estimation problem. The algorithm can find optimal solution in various settings, and also provide an extension for the case of cluster potentials.

Johnson et al. [2007] propose Lagrangian relaxation-based iterative algorithm that combines dynamic programming on thin graphs and convex optimization joining them. They describe the optimality conditions and suggest a multiscale relaxation scheme. Komodakis et al. [2007] also give a Lagrangian relaxation formulation when the problem is decomposed into trees and a subgradient method for the dual problem.

There are methods that exploit problem-specific assumptions on the potential functions or the graph structure. Kolmogorov and Zabih [2004] characterize the class of problems where the min-cut algorithm can be used to exactly solve the problem. Bayati et al. [2005] show that both the max-product algorithm and the linear programming approach find the exact MAP configuration for the maximum weight matching in bi-partite graphs.

2.6 Mean-field theory and sum-product algorithm

When MAP estimation needs to be solved for a tree distribution, it suffices to use the max-product algorithm as we have seen in previous sections. We have also seen that there exist MAP estimation methods such as tree-reweighted max-product algorithm [Wainwright et al., 2005], TRW-S [Kolmogorov, 2006], and MPLP [Globerson and Jaakkola, 2007a] that guarantee finding an upper bound of the MAP probability (or a lower bound of the minimum energy). In this section, we briefly introduce the mean field theory and the sum-product algorithm. Although they are not originally meant to be used for solving the MAP estimation problem in general, they are often used as heuristic MAP estimation methods, and have interesting theoretical connections to the MAP estimation problem under certain circumstances.

2.6.1 Mean field theory

Mean field theory (MFT) refers to approaches that attempt to compute the partition function or the marginals by making certain assumptions on the distributions. This becomes evident from the variational perspective [Wainwright and Jordan, 2003], where mean field theory is regarded as solving the inference problem for tractable distributions instead of the original distribution. For example, the problem of computing marginals can be formulated as an optimization problem in the variational perspective. Although such an optimization problem is hard to solve in general, the MFT will solve the problem for a class of tractable distribution which simplifies the entropy term in the objective function and also the constraint set representing the marginal polytope [Wainwright and Jordan, 2003]. Tractable distributions may include tree distributions, or completely disconnected distribution. For the latter case, which is also called naive mean field, the approximate distribution $q(\mathbf{x})$ can be rewritten in the form:

$$p(\mathbf{x}) \approx q(\mathbf{x}) = \prod_{i=1}^{n} q_i(x_i), \tag{2.31}$$

that is, every random variable is assumed to be independent from one another. The iterative update rule called the naive mean field updates or self-consistent mean field updates [Koehl and Delarue, 1996, Mendes et al., 1999] are designed to find a stationary point of the optimization problem based on this distributional assumption. When the distribution considered in fact follows the assumption, and if the marginals are exactly computed by the mean field updates, it is obvious that finding a maximizer of each singleton marginals will result in a MAP configuration. The estimated partition function is always guaranteed to be a lower bound of the exact partition function.

The MFT does not guarantee correct computation of marginals or the partition function in general. It is also known that the solution from the MFT depends on the initial condition because of the nonconvexity of the optimization problem [Weiss, 2001]. However, the mean field approximation tends to be exact under certain conditions as the number of variables approaches infinity [Baxter, 1982]. In practice, the MFT can be an appealing choice as an approximate inference method because it is simple and fast, but max-product and sum-product algorithms outperform the MFT empirically [Weiss, 2001, Yanover and Weiss, 2002].

2.6.2 Sum-product algorithm

The sum-product algorithm, also known as belief propagation, is another approximate inference method. It is an equivalent of the max-product algorithm for marginals computation. For tree distributions, the sum-product algorithm is guaranteed to converge in a finite number of iterations and compute correct marginals. However, for cyclic distributions, neither convergence nor correctness of marginals are guaranteed.

The sum-product algorithm can also be interpreted in the variational perspective. Whereas the mean field theory restricts the class of distributions in solving the optimization problem, the sum-product algorithm can be derived from relaxing the constraints of the problem (that is, the marginal polytope [Wainwright and Jordan, 2003]) and replacing the entropy term with the Bethe entropy approximation [Yedidia et al., 2001]. Therefore, a fixed point of the sum-product algorithm corresponds to a stationary point of the Bethe variational problem. On the other hand, when the sum-product algorithm is applied to cyclic distributions, the fixed point is mapped to pseudo-marginals into which the original distribution can be factorized. These pseudo-marginals may be inconsistent, that is, they are not guaranteed to be true marginals.

An interesting connection between the sum-product algorithm and the max-product algorithm appears in what is called the zero-temperature limits [Wainwright and Jordan, 2003]. It can be shown that the MAP estimation problem is equivalent to the computation of the scaled log partition function where the exponential parameters defining the distribution are scaled by a positive constant tending to positive infinity. Intuitively, this is because the scaling will put more emphasis on the parameters that correspond to the MAP configuration. Therefore, only the MAP configuration will retain probability mass at infinity.

The connection between the two problems suggests that the mean parameters which are the solution of the variational optimization problem for the partition function problem can produce a better approximate solution for the MAP estimation problem as the positive scaling factor increases [Wainwright and Jordan, 2003]. It is in fact found that similar ideas are the basis for simulated annealing [Kirkpatrick et al., 1983].

Chapter 3

Rotamer Optimization for Protein Design through MAP Estimation and Problem-Size Reduction

3.1 Overview of the method

In this section, we present an overview of BroMAP in a top-down manner. We start with a brief description of the branch-and-bound method as the framework of BroMAP. Then, the pruning scheme used by BroMAP is discussed in more detail.

3.1.1 Branch-and-bound framework

Figure 3-1 shows an overview of BroMAP. It is organized at the top level as a branch-and-bound method (BnB), a general problem-solving technique particularly effective for combinatorial problems [Nemhauser and Wolsey, 1988]. The basic idea of BnB is to partition the original problem recursively and solve these smaller subproblems. In the resulting search tree, each subproblem is another instance of the GMEC problem, with a different number of rotamers or residue positions from the original problem at the root node.

BnB solves the GMEC problem as a kind of tree search problem. It maintains

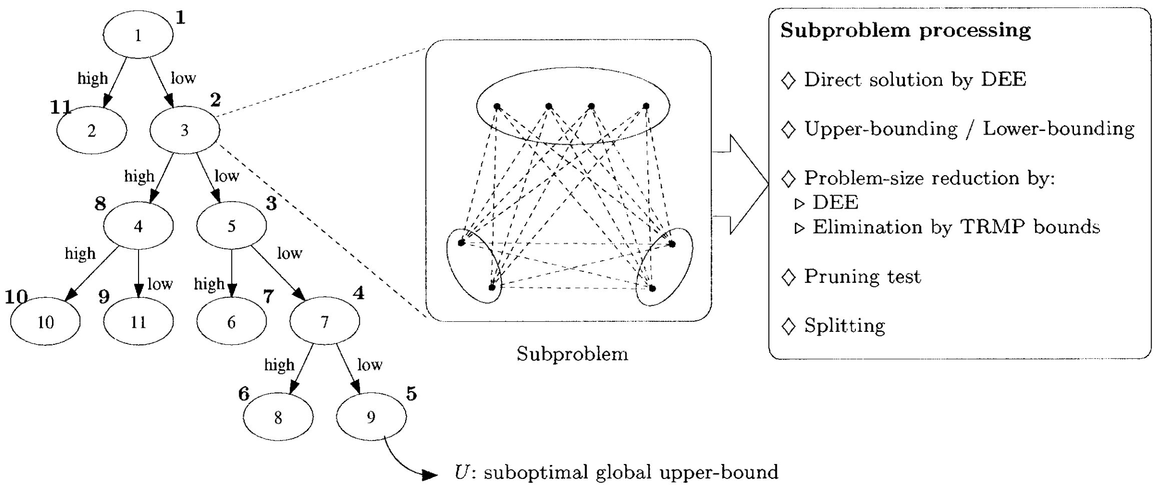


Figure 3-1: Top branch-and-bound framework of BroMAP. In the search tree, node numbers (inside the ellipses) correspond to the order of subproblem creation. Numbers shown next to ellipses represent the order of node expansion. Labels "low" and "high" marked on the branches indicate the types of child subproblems. As shown by the diagram in the middle, each subproblem is another instance of the GMEC problem; the ellipses represent the residue positions in the subproblem, and the filled dots represent available rotamer choices at each position. The lines connecting rotamers at different positions represent possible interactions between pairs of rotamers. The text box on the right side lists types of computations executed when a node is expanded.

a global upper-bound U, which is the energy of the best conformation found so far. The initial value of U is set to the energy of an arbitrary conformation. BroMAP can be recursively described as follows:

1. Select a subproblem from the queue.
2. Can the subproblem be fully solved within limited time and memory? If so,
   (a) compute the minimum energy;
   (b) set U to the minimum energy if it is less than U;
   (c) return to step 1.
3. Compute a lower bound and an upper bound on the minimum energy for this subproblem. If the upper bound is less than U, set U to the upper bound.
4. If the lower bound exceeds the current global upper-bound U, then discard (prune) this subproblem and return to step 1.
5. When possible, exclude ineligible conformations from the search space.
6. Pick one residue and split its rotamers into two groups; define two child subproblems based on this split (see Figure 3-2).
7. Add the child subproblems to the queue and return to step 1.

A node is said to be "expanded" (i.e. processed) by steps 2 to 7. This description leaves many details unspecified: how to attempt solutions, how to obtain bounds, how to identify ineligible conformations, how to choose the residue and rotamers for the node split, and what order to solve the subproblems. We provide these details in the subsequent sections.

The key advantage of BnB over naive enumeration-based methods comes from being able to approximately solve subproblems, that is, to obtain bounds on the answer that allow many subproblems to be pruned, thus avoiding exploration of the entire solution space. If the bounds are weak, BnB may end up generating too many

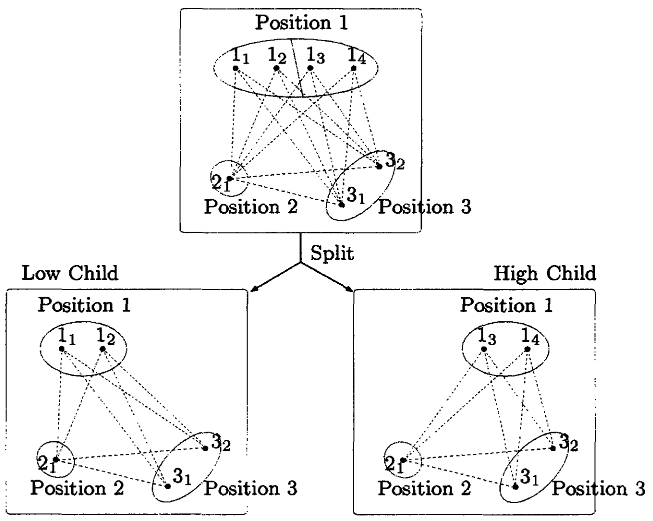


Figure 3-2: Splitting a subproblem. Rotamers at a position are divided into two groups and each child of the subproblem takes only one group of rotamers.

subproblems to be effective. The purpose of branching in a BnB method is to reduce the size of the subproblems so that they can be either solved or pruned effectively with limited resources.

In our BnB formulation, the branching rule (splitting the rotamers of a residue) only brings about a modest reduction in the search space of each child subproblem compared to its parent subproblem. Furthermore, there is no net reduction in the total search space when one considers both children. A critical component of our approach is to reduce the size of the total search space, by eliminating ineligible conformations, before splitting. This is in the spirit of the dead-end elimination algorithm or "branch-and-terminate" [Gordon and Mayo, 1999] but employing additional elimination by our new lower bounds.

3.1.2 Solving subproblems

There are two well-known approaches to solving the GMEC problem exactly. One is DEE [Desmet et al., 1992, Goldstein, 1994, Pierce et al., 2000] and the other is integer linear programming (ILP) [Nemhauser and Wolsey, 1988]. Both of these methods are guaranteed to solve the GMEC problem given unbounded resources but have unpredictable running times as a function of the problem size.

DEE is an iterative method that eliminates a non-GMEC rotamer by comparing its energetics with those of other rotamers at the same position. The same rules are also applied to eliminate rotamer pairs. When a rotamer can be eliminated from consideration, this can be represented by reducing the set of rotamers at a residue position. Eliminated rotamer pairs, on the other hand, are tracked via "pair flags", which indicate ineligible assignments for pairs of positions. When the numerical properties of the energy terms are favorable or when the problem size is relatively small, DEE successfully eliminates many non-GMEC rotamers or rotamer pairs so that the GMEC can be easily found from the remaining small conformational space. In general, one needs to perform a systematic search of the remaining conformational space; the A^* heuristic search algorithm [Hart et al., 1968] is usually used for this purpose. However, DEE may fail to reduce the size of the conformational space to the point where it is practical to search for the GMEC using A^*. This is what motivates our BnB approach.

ILP is a popular approach to solving combinatorial optimization problems but we have found that direct application of general ILP solvers to protein design problems is generally impractical (see Appendix B). Furthermore, as we discuss below, DEE has the additional advantage of reducing the size of the conformational space at each subproblem, even when it fails to completely solve the subproblem. Therefore, we have used a DEE-based solver as our method for solving subproblems.

3.1.3 Bounding subproblems

In addition to completely solving subproblems, we also need a way of obtaining lower bounds to prune nodes more efficiently. The classical approach for obtaining bounds for a combinatorial optimization problem is via the relaxation to linear programming (LP) after formulating the problem as ILP. For example, we obtain LP by treating the integer-valued variables in the ILP formulation of the GMEC problem, i.e. (B.1) – (B.6) of Appendix B, as real. Although LP problems are solvable in polynomial time, it is still the case that the LP problems resulting from the relaxation of typical protein design problems are often too large and thus require impractical amounts of computing time and memory.

The less expensive lower-bounding method that we use in this work is the tree-reweighted max-product algorithm (TRMP) [Wainwright et al., 2005], which was introduced in Section 2.4.2. TRMP lower bounds are known to be no better than the LP lower bounds, and there are no guarantees of how close to the LP bound a TRMP bound will be. However, the relatively low computational cost and its good performance in practice makes TRMP an excellent lower-bounding tool.

Another key advantage of TRMP is that, like DEE, it can be used to compute lower-bounds for parts of the conformational space efficiently and to eliminate them as discussed below.

On the other hand, upper bounds are also obtained by TRMP for the subproblems that are not exactly solved. This is based on a heuristic use of TRMP, but often produces stronger upper bounds than random sampling of conformations. We present the details on upper-bounding by TRMP later in the paper.

3.1.4 Reducing subproblem size

As we mentioned above, a critical component of our BnB methodology is that we attempt to reduce the size of the search space for each subproblem by removing ineligible conformations. Smaller subproblems are easier to solve and to bound. We use two techniques to accomplish this: DEE, as discussed above, and elimination by lower bounds. The latter is illustrated in Figure 3-3 and discussed below.

For each rotamer r at an arbitrary position i, we can think of an assignment of rotamers in other positions such that no other assignment can give a lower conformational energy when position i is fixed to r. We call the energy corresponding to such an assignment the minimum conformational energy of i_r. Similarly, we can define the minimum conformational energy for an arbitrary pair of rotamers (i_r, j_s) such that $i \neq j$.

Suppose we know a lower-bound $L(i_r)$ of the minimum conformational energy of i_r and a global upper-bound U such that $L(i_r) > U$. Then, rotamer i_r can be eliminated from the subproblem without affecting whether the subproblem is prunable

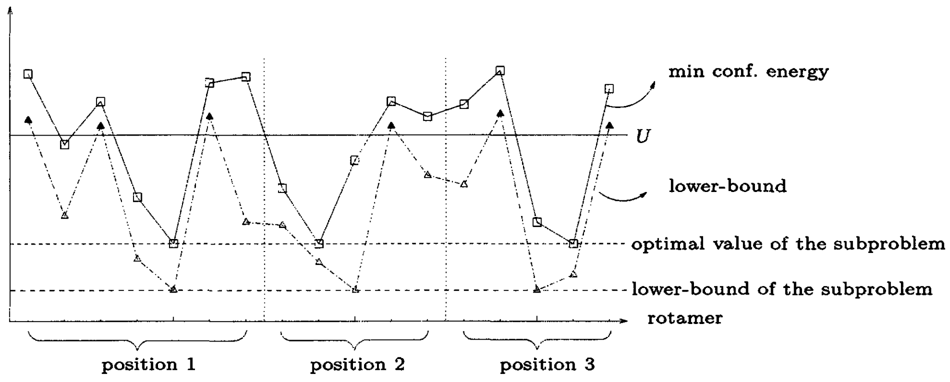
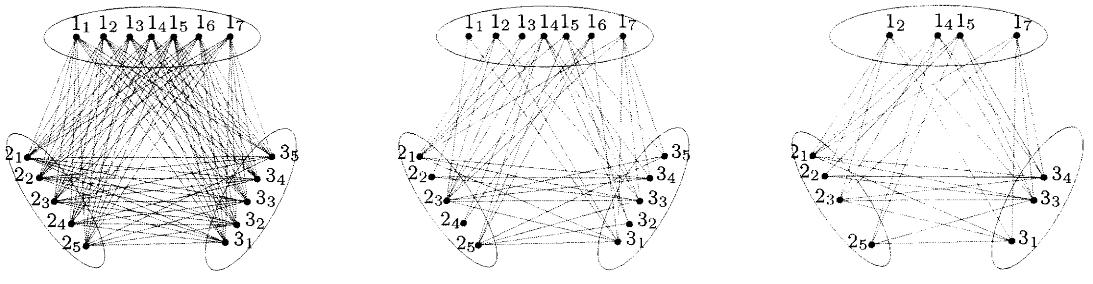


Figure 3-3: Elimination by rotamer lower bounds. The x-axis lists all rotamers of the subproblem in an arbitrary order. The vertical dotted lines indicate division of rotamers by positions they belong to. Two types of y-values are plotted for each rotamer i_r: (1) minimum energy that a conformation including i_r can have, (2) a lower bound of (1) obtained by a lower-bounding method. Three horizontal lines are also depicted, each representing (a) an upper bound U, (b) the optimal value of the subproblem, (c) a lower bound of (b) obtained from the same lower-bounding method. Rotamers that can be eliminated by comparison against U are indicated by filled triangles.

or not. Similarly, if we have a lower bound of the minimum conformational energy of a rotamer pair greater than U, the rotamer pair can also be eliminated. Figure 3-4 illustrates the problem-size reduction by elimination of rotamers and rotamer pairs.

The problem is obtaining useful lower bounds for each rotamer or rotamer pair. If we use LP relaxation, we would need to solve LP problems as many times as the number of rotamers or rotamer pairs, and each LP problem can be still very large. A more practical solution follows from the theoretical properties of TRMP, which allow us to obtain the lower bounds for all rotamers and rotamer pairs in one TRMP convergence plus post-processing time at most square of the problem size. We will discuss how we can obtain these lower bounds using TRMP later in this chapter.

When a rotamer pair is eliminated by a TRMP lower bound, we mark the rotamer pair with a pair flag, as done in DEE. However, such a pair flag is more general than the pair flags used in conventional DEE since the elimination is done relative to the current global upper-bound U. Thus, it is possible for TRMP to flag rotamer pairs belonging to the minimum energy conformation of the subproblem in case the optimal value of the subproblem is greater than U. When this happens, the optimal value



(a) Original subproblem. (b) Rotamer-pair elimination. (c) Rotamer elimination.

Figure 3-4: Reduction by elimination of rotamers and rotamer pairs. While elimination of rotamers brings explicit reduction of the problem size, elimination of rotamer pairs will be implicitly represented by pair flags. Rotamer eliminations in (c) were made consistent with bounds of Figure 3-3.

of the subproblem after the elimination can be *greater* than before the elimination. However, if the optimal value is less than equal to U, elimination by lower bounds is guaranteed to produce reduced subproblems with unchanged optimal value.

If enough pairs are eliminated by TRMP lower bounding, it may be that some positions may not have any remaining valid assignments. In this situation, the whole subproblem is infeasible and can be pruned.

Conventional DEE never flags rotamer pairs that belong to the minimum energy conformation. Therefore, the interaction of DEE with these general pair flags should be carefully considered to avoid illegal elimination by DEE. In our work, this is done by numerically enforcing the pair flags, that is, by replacing the pair flags with very large (artificial) pairwise energies. This guarantees correct elimination by DEE conditions based on energy comparison (e.g. Goldstein's conditions). Meanwhile, when logical elimination is attempted (e.g. logical singles-pairs elimination or unification), general pair flags are used as if they are conventional pair flags.

Note that we use elimination by lower bounds together with the modified DEE in each node of the search tree. In a previous work [Gordon and Mayo, 1999], lower bounds were used in the BnB framework to "terminate" singles, but DEE is only used as a preprocessing procedure before applying the BnB method. In another work [Gordon et al., 2003], elimination by lower bounds was applied in conjunction with DEE to the whole problem, but no branching was used. The lower bounds used there were also computed differently, by fixing conformations for a subset of positions and finding minimum values over decomposed sets of positions.

3.1.5 Subproblem splitting and selection

Our strategy of subproblem selection is depth-first search (DFS), where one selects the deepest subproblem to expand, breaking ties by choosing the node with the smallest lower bound. The goal is to first find a good upper-bound by following DFS through the children with the lowest bounds, then to prune the remaining subproblems using that upper-bound. To implement this strategy, we need to split subproblems so that they have substantially different lower bounds.

As discussed above, we can compute inexpensive lower bounds for individual rotamers by TRMP. Therefore, we can split a subproblem by dividing rotamers of a selected position into two groups according to their rotamer lower bounds, so that the maximum rotamer lower bound of one group is less than or equal to the minimum rotamer lower bound of the other group. We call the child from the former group "the low child" and the other as "the high child". The low child is very likely to have an optimal value less than that of the high child. A splitting position is selected so that difference between maximum and minimum rotamer lower bounds is large. This splitting scheme will also tend to make the high child easier to prune than the low child.

The leftmost diagram in Figure 3-1 illustrates our subproblem selection strategy. We can see that the tree first grows along the line of low-subproblems then the high-subproblems are traversed. We call the DFS along all low-branches until the first leaf node is reached as "the first depth-first dive". If the splitting is successful and non-optimal nodes are pruned effectively, the search tree should be highly skewed toward low-branches.

3.2 Bounding the GMEC energy with TRMP

We also make heuristic use of TRMP to obtain upper bounds for the GMEC energy. At convergence of TRMP, we occasionally find an exact MAP configuration. TRMP provides an easy evaluation condition called *optimum specification* (OS) criterion such that an assignment is guaranteed to be a MAP configuration if it satisfies the OS criterion. However, such an assignment may not exist for a given reparameterization or it could be computationally expensive to find. Therefore, in our upper bounding, instead of trying to find an assignment that satisfies the OS criterion, we simply find an assignment that maximizes the tree distribution for some star $S \in \mathcal{S}$ at TRMP convergence. How to find such an assignment can be found in Wainwright et al. [2004] Another possible upper-bounding method is to randomly pick a maximizer for each singleton max-marginals at TRMP convergence regardless of the trees. Although neither of these procedures guarantees the quality of the upper bounds, the resulting upper bounds are empirically close to the optimal values. The procedures can be repeated for different trees or different random selection of maximizers to improve the upper bounds.

A lower bound for the GMEC energy $\min_{\mathbf{x}} e(\mathbf{x})$ can be easily obtained at the convergence of TRMP with the following lemma:

Lemma 1. *When ν and ν_c of (2.30) in normal form satisfy tree-consistency condition, the MAP probability is upper bounded by*

$$\max_{\mathbf{x}} p(\mathbf{x}) \leq \nu_c. \tag{3.1}$$

Therefore, the GMEC energy $\min_{\mathbf{x}} e(\mathbf{x})$ is lower bounded by $\min_{\mathbf{x}} e(\mathbf{x}) \geq -\ln Z - \ln \nu_c$ from (2.3). Note that Lemma 1 is true not only for star covers but for general tree covers.

Example 3. *To upper bound $\max_{\mathbf{x}} p(\mathbf{x})$ using Lemma 1 and the pseudo-max-marginals given in Example 2, we first need to normalize pairwise pseudo-max-marginals. Since the maximum value of $\hat{\nu}_{ij}(x_i, x_j)$ for all (i, j) are 8, normalized pairwise pseudo-max-*

marginals are as follows:

$$\nu_{ij}(x_i, x_j) = \begin{cases} 1/8 & \textit{if } x_i = x_j \\ 1 & \textit{otherwise} \end{cases} \quad \textit{for all } (i,j) \in \{(1,2),(2,3),(3,1)\}. \qquad (3.2)$$

Single pseudo-max-marginals are already in a normal form. These lead to constant $\nu_c = 64/98$. *Therefore, the upper bound of the MAP probability is* $64/98$. *It is easy to see* $\max_{\mathbf{x}} p(\mathbf{x})$ *is equal to* $16/98$ *attained by any of* $(x_1, x_2, x_3) = (0,0,1)$, $(0,1,0)$, *etc. The upper bound of the MAP probability (thereby the resulting lower bound of the GMEC energy) is not tight in this example, but the quality of bounds from Lemma 1 can be stronger depending on pseudo-max-marginals obtained from TRMP. In this example, on the other hand, a tight lower-bound of* $p(\mathbf{x})$ *(therefore a tight upper-bound of the GMEC energy) is easily obtained by finding a MAP assignment for any of the trees in* $\mathcal{T}$. *For example,* $(x_1, x_2, x_3) = (0,1,0)$ *is a MAP assignment for tree distribution* $p^1(\mathbf{x})$, *and also for* $p(\mathbf{x})$. □

3.3 Elimination by TRMP lower bounds

We can exploit the tree-consistency of ν at TRMP convergence in computing various lower bounds for a set of conformations. If a lower bound greater than a global upper-bound U is obtained, we can eliminate corresponding conformations from the subproblem while conserving the inequality relation between the minimum energy of the subproblem and U. We make a more precise argument for what we call *rotamer-pair elimination* and *rotamer elimination* as follows. Let $\tilde{P}$ be the set of flagged rotamer pairs in the subproblem of our interest. Then, given conformational space $\mathcal{X}$, we define $\mathcal{L}(\mathcal{X}, \tilde{P})$ as the set of all legal conformations containing no flagged rotamer pairs.

1. *rotamer-pair elimination*: suppose we have a lower-bound $LB(\zeta_r, \eta_s)$ of the minimum conformational energy for $\{\mathbf{x} | (x_\zeta, x_\eta) = (r,s)\}$, the set of all conformations including (ζ_r, η_s), such that $\min_{\{\mathbf{x}|(x_\zeta,x_\eta)=(r,s)\}} e(\mathbf{x}) \geq LB(\zeta_r, \eta_s) > U$.

Elimination of (ζ_r, η_s) can be represented by the set of pair-flags $\tilde{P}' = \tilde{P} \cup (\zeta_r, \eta_s)$. We know $\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P}')} e(\mathbf{x})$ is prunable (that is, the subproblem can be deleted from the branch-and-bound tree) if and only if $\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P})} e(\mathbf{x})$ is prunable. Therefore, we use $\tilde{P}'$ as the updated set of pair flags.

2. *rotamer elimination*: suppose we have a lower-bound $LB(\zeta_r)$ of the minimum conformational energy for $\{\mathbf{x} | x_\zeta = r\}$, the set of all conformations including ζ_r, such that $\min_{\{\mathbf{x} | x_\zeta = r\}} e(\mathbf{x}) \geq LB(\zeta_r) > U$. Elimination of ζ_r can be represented by the set of pair-flags $\tilde{P}' = \tilde{P} \cup \{(\zeta_r, j_s) | s \in R_j, j \in \mathcal{V}, j \neq \zeta\}$, which includes all rotamer pairs stemming from ζ_r. Again, we know $\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P}')} e(\mathbf{x})$ is prunable if and only if $\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P})} e(\mathbf{x})$ is prunable. Therefore, we use $\tilde{P}'$ as the updated set of pair flags.

In both cases, the optimal value of $\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P})} e(\mathbf{x})$ does not change if $\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P})} e(\mathbf{x}) \leq U$.

The lower-bounds $LB(\zeta_r)$ and $LB(\zeta_r, \eta_s)$ can be, for example, obtained by directly solving an LP relaxation of the ILP given in Appendix B. However, solving LP may not be practical when the problem size is large. In addition, solving LP for every rotamer or rotamer pair will multiply the lower-bounding time by the number of rotamers or rotamer pairs. Here, we use upper-bounding inequalities for the singleton and pairwise max-marginals to obtain lower bounds for minimum conformational energies of rotamers and rotamer pairs. Such lower bounds are at best as tight as the bounds from solving the LP discussed in Appendix B [Wainwright et al., 2005], but requires computation time for one TRMP run until convergence (no guaranteed time bound) plus post-processing time at most cubic of the problem size. The rest of this section explains how we can efficiently compute the rotamer and rotamer-pair lower bounds.

We have the following lemma on upper-bounding the singleton max-marginals:

Lemma 2. *When ν and ν_c of (2.30) in a normal form satisfy tree-consistency con-*

dition, it is true for all $r \in R_i$, $\zeta \in \mathcal{V}$ that

$$\max_{\{\mathbf{x}|x_\zeta=r\}} p(\mathbf{x}) \le \nu_c \nu_\zeta(r)^{\rho_\zeta}. \tag{3.3}$$

Example 4. *From Lemma 2 and the normalized pseudo-max-marginals given in Example 3, we find an upper bound for the maximum probability of $p(\mathbf{x})$ when $x_1 = 0$ as $\nu_c\nu_1(0)^{1/3} = 64/98 \times 1^{1/3}$. The bound is not tight because $\max_{\{\mathbf{x}|x_1=0\}} p(\mathbf{x}) = 16/98$, but the tightness may change depending on the pseudo-max-marginals from TRMP. Even when the resulting bound is not tight, it could be still strong enough to eliminate the corresponding rotamer through comparison against a global upper-bound U.* □

Lemma 2 combined with (2.3) provide a rotamer lower-bound $LB(\zeta_r)$ for each $r \in R_\zeta$ and $\zeta \in \mathcal{V}$ as $\min_{\{\mathbf{x}|x_\zeta=r\}} e(\mathbf{x}) \ge LB(\zeta_r) = -\ln Z - \ln \nu_c - \rho_\zeta \ln \nu_\zeta(r)$.

To upper bound the pairwise max-marginals, we use the general inequality

$$\max_{\{\mathbf{x}|x_\zeta=r,x_\eta=s\}} p(\mathbf{x}) \le \nu_c \prod_{S\in\mathcal{S}} \left[\max_{\{\mathbf{x}|x_\zeta=r,x_\eta=s\}} p^S(\mathbf{x}) \right]^{\rho(S)}. \tag{3.4}$$

The maximization problem $\max_{\{\mathbf{x}|x_\zeta=r,x_\eta=s\}} p^S(\mathbf{x})$ can be easily solved using the following lemma:

Lemma 3. *When ν and ν_c of (2.30) in a normal form satisfy the tree-consistency condition,*

$$\max_{\{\mathbf{x}|x_\zeta=r,x_\eta=s\}} p^S(\mathbf{x}) = \begin{cases} 1 & \text{if } \zeta,\eta \notin \mathcal{V}(S) \\ \nu_\zeta(r) & \text{if } \zeta \in \mathcal{V}(S) \text{ and } \eta \notin \mathcal{V}(S) \\ \nu_{\zeta\eta}(r,s) & \text{if } (\zeta,\eta) \in \mathcal{E}(S) \\ \max_{x_\xi \in R_\xi} \frac{\nu_{\xi\zeta}(x_\xi,r)\nu_{\xi\eta}(x_\xi,s)}{\nu_\xi(x_\xi)} & \text{else (let } \xi = \gamma(S)) \end{cases} \tag{3.5}$$

Example 5. *Let us bound $\max_{\{\mathbf{x}|(x_1,x_2)=(0,0)\}} p(\mathbf{x})$ using the normalized pseudo-max-marginals given in Example 3. As discussed above, we have to solve a maximization problem for each star:*

1. *$p^1(\mathbf{x})$ and $p^3(\mathbf{x})$ (Figure 2-2(b) and 2-2(d)): this corresponds to the third case of (3.5). Therefore,* $\max_{\{\mathbf{x}|(x_1,x_2)=(0,0)\}} p^1(\mathbf{x}) = \max_{\{\mathbf{x}|(x_1,x_2)=(0,0)\}} p^3(\mathbf{x}) = \nu_{12}(0,0) = 1/8$.

2. *$p^2(\mathbf{x})$ (Figure 2-2(c)): this corresponds to the fourth case of (3.5). Therefore,*

$$\max_{\{\mathbf{x}|(x_1,x_2)=(0,0)\}} p^2(\mathbf{x}) = \max_{x_3} \frac{\nu_{3,1}(x_3,0)\nu_{3,2}(x_3,0)}{\nu_3(x_3)} = 1. \tag{3.6}$$

By combining the above results in (3.4), we obtain

$$\max_{\{\mathbf{x}|(x_1,x_2)=(0,0)\}} p(\mathbf{x}) \leq (64/98) \times (1/8)^{1/3} \times (1/8)^{1/3} \times 1^{1/3} = 16/98. \tag{3.7}$$

This bound is tight from $\max_{\{\mathbf{x}|(x_1,x_2)=(0,0)\}} p(\mathbf{x}) = 16/98$ attained by $x_3 = 1$. Note that the same pseudo-max-marginals that yielded weak upper bounds in Examples 3 and 4, led to a tight upper bound for the rotamer pair, a more constrained bounding problem. □

$LB(\zeta_r, \eta_s)$, a lower bound for the minimum conformation energy of rotamer-pair (ζ_r, η_s), is given by $LB(\zeta_r, \eta_s) = -\ln Z - \ln \nu_c - \sum_{S \in \mathcal{S}} \rho_S \ln \max_{\{\mathbf{x}|x_\zeta=r, x_\eta=s\}} p^S(\mathbf{x})$. Note that there can be at most $\mathcal{O}(n)$ stars that correspond to the fourth case of (3.5) for each position pair (ζ, η). If we let n_{rot} be the average number of rotamers per position, the maximization problem corresponding to the fourth case of (3.5) requires $\mathcal{O}(n_{rot})$ operations. Therefore, it will take $\mathcal{O}(n_{rot}n)$ post-processing operations to compute an upper bound for each rotamer pair using Lemma 3.

In computing the rotamer lower bound for a rotamer ζ_r, we can also use pair-flags information to obtain a lower bound, $LB'(\zeta_r)$, for the constrained problem $\min_{\{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P})|x_\zeta=r\}} e(\mathbf{x})$. If we have $LB'(\zeta_r) > U$, then conformations, $\{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P})|x_\zeta = r\}$ can be excluded from the search space. This is equivalent to eliminating rotamer ζ_r because all conformations containing $x_\zeta = r$ are in effect excluded. Computing $LB'(\zeta_r)$ will take additional polynomial time compared to $LB(\zeta_r)$, but it is particularly advantageous to leverage the pair flags when there exist a large number of flagged rotamer pairs. We used a simple search-based method to compute $LB'(\zeta_r)$

as follows; we let $\hat{p} = \nu_c \prod_{S \in \mathcal{S}} \left[\max_{\{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P}) | x_\zeta = r\}} p^S(\mathbf{x}) \right]^{\rho(S)}$ for tree-consistent ν in a normal form. Then, it is easy to see $\hat{p} \geq \max_{\{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P}) | x_\zeta = r\}} p(\mathbf{x})$. If we use a naive search, it will take $\mathcal{O}(n_{rot}^2 n)$ post-processing comparison operations to solve $\max_{\{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P}) | x_\zeta = r\}} p^S(\mathbf{x})$. Therefore, it takes $\mathcal{O}(n_{rot}^2 n^2)$ post-processing time to exactly compute $\hat{p}$. Finally, the rotamer lower bound is computed as $LB'(\zeta_r) = -\ln Z - \ln \hat{p}$.

Chapter 4

Problem-size reduction and lower-bounding through problem-size reduction

In this chapter, we describe three different techniques that can be used in bounding or pruning a subproblem in branch-and-bound. We first describe the notion of general pair flags that are used for implicit exclusion of a subset of conformational space. Then, we describe rotamer contraction and edge deletion, which simplify problems by aggregating rotamers or edges but result in decreases in optimal values. We also describe a dead-end elimination algorithm modified to be used with general pair flags. Finally, we provide preliminary results on using these techniques together with elimination by TRMP lower bounds in branch-and-bound.

4.1 General pair-flags

Pair-flags are used in DEE to mark a pair of rotamers from two different positions that cannot be a part of the GMEC. For example, if the pair-flag for rotamer-pair (i_r, j_s) is set, none of the conformations in $\mathcal{Z} = \{\mathbf{x} \in \mathcal{X} | (x_i, x_j) = (r, s)\}$ can be in the GMEC. Then, the GMEC problem can be solved over the conformation space $\mathcal{X} \backslash \mathcal{Z}$ instead of $\mathcal{X}$ to obtain the same GMEC. Therefore, the use of pair-flags keeps track of the constrained search space.

In this work, we use *general pair-flags*, which will be defined more generally than the conventional pair-flags to facilitate the comparison between the GMEC energy and an arbitrary upper-bound. For each GMEC problem $\min_{\mathbf{x}} e(\mathbf{x})$, we simply assume that we are given a set of rotamer-pairs $\tilde{P}$ that should be excluded from the search for the GMEC. Although DEE never flags a rotamer-pair corresponding to the GMEC, there is generally no such restriction on $\tilde{P}$, which may vary from an empty set to the set of all rotamer-pairs in the problem. For notational purpose, we define pair-flag functions as

$$\tilde{g}_{ij}(r,s,\tilde{P}) = \begin{cases} 1 & \text{if } (i_r, j_s) \in \tilde{P} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

and $\tilde{g}(\mathbf{x},\tilde{P}) = \sum_{i,j\in\mathcal{V},i\neq j} \tilde{g}_{ij}(x_i,x_j,\tilde{P})$. Then, any legal conformation that does not violate the pair-flag constraints will satisfy $\tilde{g}(\mathbf{x},\tilde{P}) = 0$. Therefore, given conformation space $\mathcal{X}$ and a set of pair-flags $\tilde{P}$, we define $\mathcal{L}(\mathcal{X},\tilde{P}) = \{\mathbf{x}|\tilde{g}(\mathbf{x},\tilde{P}) = 0\}$, i.e. the set of all legal conformations.

We also define the set of rotamer-pairs $P(\{e\},U)$ that can be eliminated against a global upper bound U, i.e. $P(\{e\},U) = \{(i_r,j_s) \mid \min_{\{\mathbf{x}|(x_i,x_j)=(r,s)\}} e(\mathbf{x}) > U,\ i \neq j\}$. When U is equal to $\min_{\mathbf{x}} e(\mathbf{x})$, $P(\{e\},U)$ becomes equivalent to the set of pair-flags used by DEE although DEE often only knows a subset of $P(\{e\},U)$ in the course of solving the GMEC problem.

Without any restriction on the value of U, we have the following lemma regarding minimization under pair-flag constraints:

Lemma 4. *For any $\tilde{P}$ and $\tilde{P}'$ such that $\tilde{P} \subset \tilde{P}'$ and $\tilde{P}'\backslash\tilde{P} \subset P(\{e\},U)$,*

1. *if $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x}) > U$, then $\mathcal{L}(\mathcal{X},\tilde{P}')$ is empty or $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P}')} e(\mathbf{x}) > U$.*

2. *if $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x}) \leq U$, then $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P}')} e(\mathbf{x}) = \min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x})$.*

3. *if $\mathcal{L}(\mathcal{X},\tilde{P}')$ is empty, then $\mathcal{L}(\mathcal{X},\tilde{P})$ is empty or $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x}) > U$.*

We say a subproblem $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x})$ is infeasible when $\mathcal{L}(\mathcal{X},\tilde{P})$ is empty. We can regard the optimal value of an infeasible subproblem as $+\infty$ since the subproblem can be pruned by any U. The following corollary is immediately obtained from Lemma 4:

Corollary 1. *For any $\tilde{P}$ and $\tilde{P}'$ such that $\tilde{P} \subset \tilde{P}'$ and $\tilde{P}' \backslash \tilde{P} \subset P(\{e\}, U)$, $\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P}')} e(\mathbf{x})$ is either infeasible or greater than U if and only if $\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P})} e(\mathbf{x})$ is either infeasible or greater than U.*

The implication of Corollary 1 is that given a subproblem $\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P})} e(\mathbf{x})$ in the BnB-tree and a global upper-bound U, the subproblem can be pruned in comparison with U if and only if the modified subproblem $\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P}')} e(\mathbf{x})$ can be pruned against U. In what follows, when we need to mention pair-flag information, we will implicitly assume we have some $\tilde{P}$, and use the notation $\tilde{g}(\mathbf{x})$ instead of $\tilde{g}(\mathbf{x}, \tilde{P})$ where specifying $\tilde{P}$ is not particularly necessary.

Let us define $\Gamma(i)$ as the set of vertices neighboring i in $\mathcal{G}$ for each $i \in \mathcal{V}(\mathcal{G})$. The following condition on pair-flags will be used in the following sections:

Condition 1. *For all $r \in R_i$ and $i \in \mathcal{V}$, there exists $s \in R_j$ for each $j \in \Gamma(i)$ such that $(i_r, j_s) \notin \tilde{P}$.*

Condition 1 can be maintained without loss of generality if, whenever $\tilde{P}$ is updated, we detect rotamers i_r such that $(i_r, j_s) \in \tilde{P}$ for all $s \in R_j$ for some $j \in \Gamma(i)$ and delete such i_r from the problem, which does not change the solution of the problem.

4.2 Rotamer contraction

Let ζ be the position whose rotamers we partition into a number of clusters $C_1, \ldots, C_l$, $l < |R_\zeta|$. Then, we contract all rotamers $r \in C_k$ as one rotamer-aggregate c_k. The contracted GMEC problem has a new conformation space $\mathcal{X}^{rc}$, which is the same as $\mathcal{X}$ except that R_ζ of $\mathcal{X}$ is replaced by $\{c_1, \ldots, c_l\}$. Then, we define a new energy function $e^{rc}(\mathbf{x})$ over $\mathcal{X}^{rc}$ and the set of pair-flags $\tilde{P}^{rc}$ so that the optimal value of the contracted problem $\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}^{rc}, \tilde{P}^{rc})} e^{rc}(\mathbf{x})$ is a lower-bound of $\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P})} e(\mathbf{x})$, the optimal value before the contraction. One way of choosing $e^{rc}(\mathbf{x})$ for a given clustering is given by *contract-rotamers* in Algorithm 1, where it takes the minimum of unflagged pairwise energies for each cluster; the pairwise energies are augmented by a fraction of the associated singleton energies. We use notation $e^{rc}(\mathbf{x}, \tilde{P})$ to indicate the function is also defined by $\tilde{P}$.

Algorithm 1: contract-rotamers

```
   Data: $\zeta$, $C_1, \ldots, C_l$, $\mathcal{X}$, $\{e\}$, $\tilde{P}$
   Result: $\mathcal{X}^{rc}$, $\{e^{rc}\}$, $\tilde{P}^{rc}$
 1 begin
 2     $\mathcal{X}^{rc}$ is same with $\mathcal{X}$ except $R_\zeta$ is replaced with $\{c_1, \ldots, c_l\}$
 3     $\tilde{P}^{rc} \leftarrow \tilde{P} \backslash \{(\zeta_r, j_s), j \in \Gamma(\zeta)\}$
 4     foreach $C_k$, $k = 1, \ldots, l$ do
 5         foreach $s \in R_j$, $j \in \Gamma(\zeta)$ do
 6             $e^{rc}_{\zeta j}(c_k, s, \tilde{P}) \leftarrow \min_{\{r \in A, (\zeta_r, j_s) \notin \tilde{P}\}} e_{\zeta j}(r, s) + \frac{e_\zeta(r)}{|\Gamma(\zeta)|}$,
 7             if $(\zeta_r, j_s) \in \tilde{P}$ for all $r \in C_k$ then $\tilde{P}^{rc} \leftarrow \tilde{P}^{rc} \cup (\zeta_{c_k}, j_s)$
 8         $e^{rc}_\zeta(c_k) \leftarrow 0$.
 9     define $e^{rc}(\mathbf{x})$ same as $e(\mathbf{x})$ for other terms
10 end
```

We have the following lemma on *contract-rotamers*:

Lemma 5. *For any given clustering of rotamers of $\zeta \in \mathcal{V}$, the following claims hold:*

1. *If $\mathcal{L}(\mathcal{X}^{rc}, \tilde{P}^{rc}) = \phi$, then $\mathcal{L}(\mathcal{X}, \tilde{P}) = \phi$.*

2. *If $\mathcal{L}(\mathcal{X}^{rc}, \tilde{P}^{rc}) \neq \phi$, then*
$\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P})} e(\mathbf{x}) \geq \min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}^{rc}, \tilde{P}^{rc})} e^{rc}(\mathbf{x}, \tilde{P}) \geq \min_{\mathbf{x} \in \mathcal{X}^{rc}} e^{rc}(\mathbf{x}, \tilde{P}) \geq \min_{\mathbf{x} \in \mathcal{X}^{rc}} e^{rc}(\mathbf{x}, \phi)$.

Lemma 5.1 suggests the contracted problem is infeasible only if the original problem is infeasible. Lemma 5.2 shows the rotamer contraction is in fact a lower-bounding operation.

A similar approach was previously used by Koster et al. [1999] for the frequency assignment problem, where frequencies belonging to the same frequency block usually take similar pair-penalty and can be aggregated easily. However, they use the lower-bounding in reverse direction where the lower-bound does not decrease by starting from the most coarsely aggregated bounding problem to finer clustering of frequencies. In large problems, it will be hard to solve the bounding problem exactly after certain number of iterations of refinements. Therefore resorting back to another lower-bounding to solve a refined problem will be necessary. In addition, by using our strategy of iteratively reducing the problem-size, reductions can be also contributed by elimination by rotamer/rotamer-pair lower-bounds, edge-deletion and DEE.

In rotamer contraction, how we cluster rotamers of position ζ determines the quality of resulting lower-bounds. Generally, finding the smallest number of clusters for a given maximum decrease in the optimal value is *NP*-hard as can be shown by reduction from the k-center problem [Fowler et al., 1981, Meggiddo and Supowit, 1984]. Our approach is a greedy scheme that keeps placing rotamers in a cluster as long as the decrease in the optimal value is less than equal to a specified amount. However, it is hard to exactly know the decrease $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x}) - \min_{\mathbf{x}\in\mathcal{L}(\mathcal{X}^{rc},\tilde{P}^{rc})} e^{rc}(\mathbf{x},\tilde{P})$. In addition, it is generally not feasible to bound the decrease since rotamer contraction may even turn an infeasible subproblem into a feasible one. We instead upper-bound $\Delta OPT^{rc} \overset{def}{=} \min_{\mathbf{x}\in\mathcal{X}} e(\mathbf{x}) - \min_{\mathbf{x}\in\mathcal{L}(\mathcal{X}^{rc},\tilde{P}^{rc})} e^{rc}(\mathbf{x},\tilde{P})$. Let

$$U^{rc}_{\Delta OPT}(\tilde{P}) = \max_{1\le k\le l}\min_{r\in C_k}\sum_{j\in\Gamma(\zeta)}\max_{\{s\in R_j|(\zeta_{c_k},j_s)\notin\tilde{P}^{rc}\}}\{e_{\zeta j}(r,s)+\frac{e_\zeta(r)}{|\Gamma(\zeta)|}-e^{rc}_{\zeta j}(c_k,s,\tilde{P})\}. \quad (4.2)$$

Then, we have the following lemma:

Lemma 6. *For any given clustering of rotamers of $\zeta\in\mathcal{V}$, we have $\Delta OPT^{rc}\le U^{rc}_{\Delta OPT}(\tilde{P})\le U^{rc}_{\Delta OPT}(\phi)$*

Note that $U^{rc}_{\Delta OPT}(\tilde{P})$ has a finite value due to Condition 1. From Lemma 6, the optimal value of the contracted problem is lower-bounded by $\min_{\mathbf{x}\in\mathcal{X}} e(\mathbf{x}) - U^{rc}_{\Delta OPT}(\tilde{P})$. $\tilde{\Delta} OPT^{rc}$ can even be negative. In fact, we can construct cases where

$$\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x}) > \min_{\mathbf{x}\in\mathcal{X}} e(\mathbf{x}) > \min_{\mathbf{x}\in\mathcal{L}(\mathcal{X}^{rc},\tilde{P}^{rc})} e^{rc}(\mathbf{x}) \quad (4.3)$$

as well as cases where

$$\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X}^{rc},\tilde{P}^{rc})} e^{rc}(\mathbf{x}) > \min_{\mathbf{x}\in\mathcal{X}} e(\mathbf{x}). \quad (4.4)$$

On the other hand, the inequality $U^{rc}_{\Delta OPT}(\tilde{P}) \le U^{rc}_{\Delta OPT}(\phi)$ suggests using pair-flags in *contract-rotamer* also gives an upper-bound less than equal to the one computed from empty pair-flag set.

Our suggested scheme *greedy-clustering* starts with an empty cluster. Then, *greedy-clustering* checks with every other rotamer whether adding the rotamer to the current cluster will increase $U^{rc}_{\Delta OPT}$ over some given threshold Δ^{rc}. The procedure is summarized in Algorithm 2. As enforced by line 8 of Algorithm 9, the upper-bound on the decrease of the optimal value for each cluster is less than Δ^{rc}. As a result, we also have $\Delta OPT^{rc} < \Delta^{rc}$. When $\min_{\mathbf{x}} e(\mathbf{x})$ is greater than U, Δ^{rc} can be safely allowed to be at most $\min_{\mathbf{x}\in\mathcal{X}} e(\mathbf{x}) - U$ or some fraction of it without affecting the prunability. Since we do not know the exact value of $\min_{\mathbf{x}\in\mathcal{X}} e(\mathbf{x})$, Δ^{rc} is heuristically set as a fraction of the difference between an upper-bound of $\min_{\mathbf{x}\in\mathcal{X}} e(\mathbf{x})$ and U. Both upper-bounds can be obtained using TRMP. On the other hand, even if we obtain an upper-bound of $\min_{\mathbf{x}\in\mathcal{X}} e(\mathbf{x})$ less than equal to U, the optimal value of $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x})$ can be still greater than U. Therefore, some arbitrary $\Delta^{rc} > 0$ can be used until we exactly solve the reduced problem.

Algorithm 2: greedy-clustering

```
  Data: $\zeta$, $\Delta^{rc}$, $\{e\}$
  Result: $\{C_i\}$
1 begin
2     $i \leftarrow 0$, $B \leftarrow R_\zeta$
3     while $B$ is not empty do
4         $i \leftarrow i+1$, $C_i \leftarrow \phi$
5         foreach $z$ in $B$ do
6             $\{\mathcal{X}^{rc}, e'(\mathbf{x})\} \leftarrow$ contract-rotamers($\zeta$, $C_i \cup \{z\}$, $\mathcal{X}$, $e(\mathbf{x})$)
7             $\Delta \leftarrow \min_{r\in C_i} \sum_{j\in\Gamma(\zeta)} \max_{s\in R_j}\{e_{\zeta j}(r,s) + \frac{e_\zeta(r)}{\Gamma(\zeta)} - e'_{\zeta j}(\alpha, s)\}$
8             if $\Delta < \Delta^{rc}$ then $C_i \leftarrow C_i \cup \{z\}$, $B \leftarrow B\backslash\{z\}$
9 end
```

There can be various ways of choosing the contraction position ζ. Since the purpose of contraction is reduction of the problem-size, ζ can be selected as follows:

- choose position that brings the largest reduction in number of rotamers, i.e. $\zeta = \arg\max_i\{(\#\text{rotamers of } i) - (\#\text{rotamers of } i \text{ after contraction})\}$.
- choose position that reduces the number of conformations most, i.e. $\zeta = \arg\max_i\{(\text{reduction in } \#\text{rotamers of } i) \times \prod_{j\neq i}(\#\text{rotamers at position } j)\}$.

- choose position that reduces the number of rotamer pairs most, i.e.
  $\zeta = \arg\max_i\{(\text{reduction in \#rotamers of } i) \times \sum_{j \in \Gamma(i)}(\text{\#rotamers at position } j)\}$.

In our experiments of Section 4.5, we used the third criterion.

4.3 Edge-deletion

In edge-deletion, we first identify a pair of positions (ζ, η) such that the deviation in $e_{\zeta\eta}(r, s)$ for all $(r, s) \in R_\zeta \times R_\eta$ is small, then set all the interaction energies of the position pair to the minimum interaction energies. That is, the new energy function $e^{ed}(\mathbf{x})$ after edge-deletion will be defined to be same as $e(\mathbf{x})$ as except:

$$e^{ed}_{\zeta\eta}(r, s, \tilde{P}) = e^{ed}_{\zeta\eta} \overset{def}{=} \min_{\{(r,s) \in R_\zeta \times R_\eta | (r,s) \notin \tilde{P}\}} e_{\zeta\eta}(r, s), \quad (r, s) \in R_\zeta \times R_\eta, \tag{4.5}$$

Then, we have

$$e^{ed}(\mathbf{x}, \tilde{P}) = \sum_{i \in \mathcal{V}} e^{ed}_i(x_i) + \sum_{(i,j) \in \mathcal{E}} e^{ed}_{ij}(x_i, x_j) = e^{ed}_{\zeta\eta} + \sum_{i \in \mathcal{V}} e^{ed}_i(x_i) + \sum_{(i,j) \in \mathcal{E} \setminus (\zeta,\eta)} e^{ed}_{ij}(x_i, x_j) \tag{4.6}$$

Therefore, we can exclude position pair (ζ, η) from further consideration. More precisely, $\mathcal{E}$ is replaced by $\mathcal{E} \setminus \{(\zeta, \eta)\}$. The same idea was employed by Xie and Sahinidis [2006] as an approximation procedure to solving the GMEC problem. Some advantages of performing edge-deletion are as follows:

- when a significant portion of edges are deleted, we may apply more direct solution techniques such as dynamic programming Xu [2005], Leaver-Fay et al. [2005].

- empirically, being able to cover the graph using a smaller number of trees is favorable for obtaining tighter lower-bounds from TRMP.

- in rotamer contraction, a smaller number of clusters can be obtained for the same Δ^{rc} since the greedy-clustering will consider fewer interactions in computing the upper-bounds on decrease of the optimal value.

Let

$$\tilde{\Delta}OPT^{ed} \overset{def}{=} \min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x}) - \min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e^{ed}(\mathbf{x},\tilde{P}), \tag{4.7}$$

and

$$U^{ed}_{\Delta OPT}(\tilde{P}) = \max_{\{(r,s)\in R_\zeta\times R_\eta|(\zeta_r,\eta_s)\notin\tilde{P}\}} e_{\zeta\eta}(r,s) - e^{ed}_{\zeta\eta}. \tag{4.8}$$

Note that $e^{ed}_{\zeta\eta}$ and $U^{ed}_{\Delta OPT}$ are always defined and finite by Assumption 1. Then, analogously to Lemma 5 and Lemma 6, the following observations can be made:

if we have $\mathcal{L}(\mathcal{X},\tilde{P}) \neq \phi$, and delete edge $(\zeta,\eta)\in\mathcal{E}$,

1. $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x}) \geq \min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e^{ed}(\mathbf{x},\tilde{P}) \geq \min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e^{ed}(\mathbf{x},\phi)$.

2. $\Delta OPT^{ed} \leq U^{ed}_{\Delta OPT}(\tilde{P}) \leq U^{ed}_{\Delta OPT}(\phi)$

The first statement above suggests using pair-flags is also advantageous for edge-deletion as it was in rotamer contraction. When we attempt to control the decrease of the optimal value from edge-deletion as done in rotamer contraction, we may allow edge-deletion only if $U^{ed}_{\Delta OPT} \leq \Delta^{ed}$ for some $\Delta^{ed} > 0$.

4.4 Dead-End elimination

We regard DEE as consisting of three main components: flagging, singles-elimination, and unification. Based on such dissection of DEE, we describe how DEE can help pruning a subproblem as a problem-size reduction tool.

4.4.1 Flagging

As mentioned in Section 4.4, there exists rich literature describing various flagging techniques. Here we assume some of the known flagging techniques are used. In general, such flagging techniques can be described as follows:

1. Flag a rotamer ζ_r (singles-flagging) only if

$$\min_{\{\mathbf{x}|x_\zeta=r\}} e(\mathbf{x}) > \min_{\{\mathbf{x}|x_\zeta=s\}} e(\mathbf{x}) \text{ for some } s\in R_\zeta\backslash r.$$

2. Flag a rotamer pair (ζ_r, η_s) (pairs-flagging) only if

$$\min_{\{\mathbf{x} | (x_\zeta, x_\eta) = (r,s)\}} e(\mathbf{x}) > \min_{\{\mathbf{x} | (x_\zeta, x_\eta) = (t,u)\}} e(\mathbf{x}) \text{ for some } (t,u) \in R_\zeta \times R_\zeta \backslash (r,s).$$

Therefore, only a non-GMEC rotamer/rotamer-pair are flagged for the GMEC problem $\min_{\mathbf{x}} e(\mathbf{x})$.

4.4.2 Singles-elimination

We refer to rotamer elimination solely based on given single/pair-flags as singles-elimination. We can eliminate a rotamer ζ_r and all its associated rotamer-pairs from the conformation space if one of the following condition is satisfied:

1. ζ_r is flagged.
2. $(\zeta_r, j_s) \in \tilde{P}$, for all $s \in R_j$ and some $j \in \Gamma(\zeta)$.

The second condition is a logical statement that ζ_r cannot be in GMEC if all rotamer-pairs (ζ_r, j_s), $s \in R_j$ are flagged. Singles-flagging is taken to be the same as singles-elimination in most literature on DEE, but we explicitly separate singles-flagging and singles-elimination here to augment singles-elimination with general pair-flags later.

4.4.3 Unification

The idea of unification is to merge two positions ζ and η into one new position $[\zeta : \eta]$. Each rotamer $t \in R_{[\zeta:\eta]}$ corresponds to a pair of rotamers $(\zeta_r, \eta_s) \in R_\zeta \times R_\eta$. The benefit of doing unification comes when there are many dead-end pairs (ζ_r, η_s) such that $(\zeta_r, \eta_s) \in \tilde{P}$, because a dead-end pair can be eliminated as a single rotamer at unification. The resulting number of rotamers at position $[\zeta : \eta]$ will be

$$|R_{[\zeta:\eta]}| = |R_\zeta||R_\eta| - |\tilde{P}(\zeta, \eta)|, \tag{4.9}$$

where $\tilde{P}(\zeta, \eta) \overset{def}{=} \{(\zeta_r, \eta_s) \mid (\zeta_r, \eta_s) \in \tilde{P}\}$. Therefore, unification is a way of exposing the pair-flag information combinatorially. More details on unification can be found from Desmet et al. [1992] and Goldstein [1994]. Although unification often brings a break-through when singles/pairs-flagging underperforms, using it too liberally may also aggravate the situation. Possible conditions to allow unification are:

- when the fraction of dead-end pairs are large, i.e. $\frac{|\tilde{P}(\zeta,\eta)|}{|R_\zeta||R_\eta|} > \tau_1$ for some threshold $\tau_1 > 0$.
- when the number of rotamers in the new position will not be large, i.e. $|R_\zeta||R_\eta| - |\tilde{P}(\zeta,\eta)| < \tau_2$ for some threshold $\tau_2 > 0$.

4.4.4 DEE with general pair-flags

General pair-flags can be exploited to facilitate the function of each reduction method. However, DEE is not guaranteed to find the optimal solution for the GMEC problem constrained by general pair-flags. Figure 4-1 shows an example where Goldstein's singles-elimination condition yields an incorrect optimal solution. In the example, without the pair-flag, the optimal solution would be $(1_2, 2_2, 3_1)$ with optimal value -3. Therefore, rotamer 1_1 could be correctly eliminated without affecting the optimal value. On the other hand, if $(1_2, 2_2)$ were flagged, then the minimization problem constrained by the pair-flag would have the optimal solution $(1_1, 2_2, 3_1)$ with the optimal value -1. Goldstein's singles-elimination condition will, however, flag rotamer 1_1 since, when the energy from the position pair $(2,3)$ is not considered, 1_1 always has larger energy contribution to the total conformation energy than 1_2, even when the pair-flag $(1_2, 2_2)$ is considered. Therefore, Goldstein's singles-elimination condition becomes incorrect when used with general pair-flags.

Our scheme for using DEE with general pair-flags is replacing the pair-flags with numerical constraints. That is, before applying DEE on the problem, we modify the problem by setting the pairwise energies to a very large value and entirely dropping the pair-flag constraints. There are three drawbacks in this scheme. One is redundant efforts spent on flagging rotamer-pairs that were already flagged. Another is that we might not be able to recover all the original pair-flags, which, however, happens only if the original problem is infeasible. Figure 4-2 shows an example where none of the

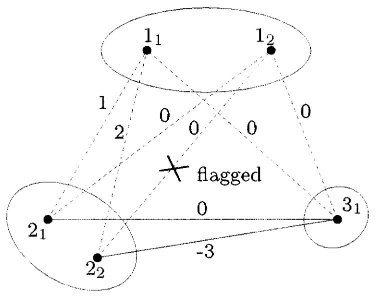


Figure 4-1: Example of incorrect rotamer-pair elimination by DEE when using general pair-flags.

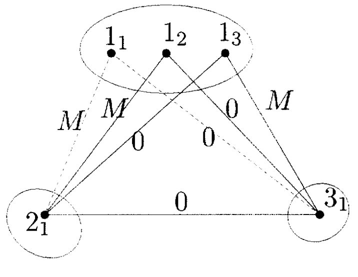


Figure 4-2: Example of pair-flags that cannot be recovered by DEE with modified energy terms. All rotamer-pairs with energy M were originally flagged, thereby making the problem infeasible. None of these rotamer-pairs will be flagged again by DEE.

originally flagged rotamer-pairs will be flagged by DEE when we replace the energy of the flagged-rotamer pair to a very large positive number M. The third problem is that the energy terms for the pairs that were flagged before DEE would be very large after DEE and this might produce a numerical instability in TRMP unless such large numbers are properly treated. In what follows, we present an adaptation of DEE to avoid the latter two problems.

Algorithm 3 shows the top framework of the modified DEE, *DEE-gp*. The routine outputs a problem $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P}')} e'(\mathbf{x})$ modified from the original problem $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x})$. The basic idea of *DEE-gp* is to apply the same reduction on $\{e'_i, e'_{ij}\}$ and $\tilde{P}'$ as performed on $\{\hat{e}_i, \hat{e}_{ij}\}$ and $\hat{P}$, where $\{e'_i, e'_{ij}\}$ and $\tilde{P}'$ are initialized by $\{e_i, e_{ij}\}$ and $\tilde{P}$ respectively. Then, the energy terms for the reduced problem are $\{e'_i, e'_{ij}\}$, and the new pair-flags are the union of $\tilde{P}'$ and the pair-flags $\hat{P}$ from DEE. In Algorithm 3, we let $\hat{D}$ a set of flagged single rotamers. We assume there exist routines *singles-pairs-flagging* that flag single rotamers and rotamer-pairs according to DEE conditions.

Algorithms used in *singles-pairs-flagging* can be arbitrary as long as they flag only non-GMEC singles and pairs. In line 4, we can terminate the elimination efforts and exit through line 13, for example, when the number of newly flagged rotamer-pairs is smaller than some threshold. Subprocedures *eliminate-singles* and *unification* can be found in Appendix B.2.

Algorithm 3: DEE-gp

```
   Data: $\mathcal{G}, \{R_i\}, \{e\}, \tilde{P}$
   Result: $\mathcal{G}', \{R_i'\}, \{e'\}, \tilde{P}'$
1  begin
2      $\mathcal{G}' \leftarrow \mathcal{G}$, $\{R_i'\} \leftarrow \{R_i\}$, $\{e'\} \leftarrow \{e\}$, $\tilde{P}' \leftarrow \tilde{P}$, $\hat{D} \leftarrow \phi$, $\hat{P} \leftarrow \phi$
3      define $\hat{e}(\mathbf{x})$ the same as $e(\mathbf{x})$ except $\hat{e}_{ij}(x_i) \leftarrow M \gg 0$ for $(x_i, x_j) \in \tilde{P}$, $(i,j) \in \mathcal{E}$
4      if *termination condition is met* then go to line 13
5      else $\{\hat{D}, \hat{P}\} \leftarrow$ *singles-pairs-flafgging*$(\{\hat{e}\}, \hat{D}, \hat{P})$
6      if *any rotamer-pairs were flagged from line 5* then
7          $\{\{R_i'\}, \{\hat{e}\}, \hat{P}, \{e'\}, \tilde{P}'\} \leftarrow$ *eliminate-singles*$(\{R_i'\}, \{\hat{e}\}, \hat{D}, \hat{P}, \{e'\}, \tilde{P}')$
8          $\hat{D} \leftarrow \phi$, go to line 4
9      else if *unification is feasible* then
10         determine unification positions $\zeta$, $\eta$
11         $\{\mathcal{G}', \{R_i'\}, \{\hat{e}\}, \hat{P}, \{e'\}, \tilde{P}'\} \leftarrow$ *unification* $(\mathcal{G}', \{R_i'\}, \{\hat{e}\}, \hat{P}, \{e'\}\tilde{P}', \zeta, \eta)$
12         go to line 4
13     $\tilde{P}' \leftarrow \tilde{P}' \cup \hat{P}$
14 end
```

For the rest of this section, we will show that *DEE-gp* is an exact problem-size reduction tool that does not change the optimal value. First, we have the following lemma regarding the replacement of pair-flags with numerical constraints in *DEE-gp*:

Lemma 7. *Let $\mathcal{L}(\mathcal{X}, \tilde{P}) \neq \phi$ and $\hat{e}(\mathbf{x})$ defined as in line 3 of Algorithm 14, where M is a large positive number satisfying $\hat{e}(\mathbf{z}) \geq M \gg \min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P})} e(\mathbf{x})$ for all $\mathbf{z}$ such that $(z_\zeta, z_\eta) \in \tilde{P}$ for some $(\zeta, \eta) \in \mathcal{E}$. Then, we have* $\arg\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P})} e(\mathbf{x}) = \arg\min_{\mathbf{x}} \hat{e}(\mathbf{x})$.

Since $\hat{e}(\mathbf{z}) \geq M + \sum_{(i,j) \neq (\zeta, \eta)} \hat{e}_{ij}(z_i, z_j) + \sum_i \hat{e}_i(z_i)$ if $(z_\zeta, z_\eta) \in \tilde{P}$ for some $(\zeta, \eta) \in \mathcal{E}$, setting $M \to \infty$ will always satisfy the condition on M in Lemma 7.

We have the following lemma on *DEE-gp*:

Lemma 8. *The following invariant holds at each step of DEE-gp after line 3 and before line 13:*

1. $\arg\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P}')} e'(\mathbf{x}) = \arg\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} \hat{e}(\mathbf{x})$ *if* $\mathcal{L}(\mathcal{X},\tilde{P}) \neq \phi$, *and* $\mathcal{L}(\mathcal{X},\tilde{P}') = \phi$ *if* $\mathcal{L}(\mathcal{X},\tilde{P}) = \phi$.

2. *for any* $\mathbf{z} \in \arg\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P}')} e'(\mathbf{x})$, *we have* $e'(\mathbf{z}) = \hat{e}(\mathbf{z})$.

The following corollary can be obtained from Lemma 8:

Corollary 2. *Suppose we obtain* $\{\mathcal{G}', \{R_i'\}, \{e'\}, \tilde{P}'\}$ *as output from DEE-gp for the input* $\{\mathcal{G}, \{R_i\}, \{e\}, \tilde{P}\}$. *Then, we have* $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P}')} e'(\mathbf{x}) = \min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x})$ *if* $\mathcal{L}(\mathcal{X},\tilde{P}) \neq \phi$, *and* $\mathcal{L}(\mathcal{X},\tilde{P}') = \phi$ *if* $\mathcal{L}(\mathcal{X},\tilde{P}) = \phi$.

4.5 Computational experiments

In our numerical experiments, a Linux workstation with a 2.2 GHz Intel Xeon processor and 3.5 Gbytes of memory was used. Table 5.1 shows 12 protein design cases used in the experiments. DEE on each case was performed with the following options: Goldstein's singles elimination, splitting with split flags ($s = 1$), Goldstein's pair elimination with one magic bullet, and unification allowing maximum 6,000 rotamers per position. E-9 finished in 4.8 hours but none of others were solved within 48 hours.

To evaluate our pruning scheme, we compared it (call it PbyR: *prune-by-reduction*) against linear programming (LP). We used subproblems of various sizes generated while solving the design cases of Table 5.1 with branch-and-bound. We used the LP formulation of Appendix B.3, and solved it with a C++ procedure using the CPLEX 8.0 library. In PbyR, we alternated rotamer contraction and edge-deletion at every iteration. At every 8th reduction, we applied DEE to see if we could solve the reduced problem or only to flag more rotamer/rotamer-pairs. (Note that we used *DEE-gp*, the DEE adapted to be compatible with general pair-flags.) We computed TRMP lower-bounds at every 24th reduction and flagged rotamers/rotamer-pairs. We allowed at most 300 reductions until we find a lower-bound greater than U or exactly solve the reduced problem. Figure 4-3 and 4-4 shows the result for the 156 subproblems remaining after excluding the subproblems that could be solved quickly by DEE alone.

Table 4.1: Test cases facts. All cases are from antigen-antibody model system. Each case repacks either the antigen protein or the antibody, or both. Each column represents (1) case name, (2) number of positions, (3) maximum number of rotamers offered at a position, (4) number of total rotamers, (5) $\sum_{i=1}^{n} \log_{10} |R_i|$, (6) case composition ('m' – #positions allowed to mutate, 'n' – #positions only wild-types are allowed, 'w' – #water molecules to be oriented at the interface). In the case names, `R` uses the standard rotamer library, and `E` multiplies each of χ_1 and χ_2 by a factor of 3 by adding $\pm 10°$. `E-1` were offered only hydrophobic residues while others were offered both hydrophobic and polar residues. All energies were calculated using the CHARMM package and the parameter set 'param22'

Case	n	$\max\|R_i\|$	$\sum\|R_i\|$	$\log_{10} conf$	Composition
R-1	34	125	1422	30.0	34 m
R-2	30	133	1350	40.2	30 m
E-1	19	617	3675	38.1	19 m
E-2	23	1370	9939	52.3	23 m
E-3	23	1320	8332	49.1	23 m
E-4	15	1361	7467	33.9	15 m
E-5	24	1344	9585	49.6	24 m
E-6	36	1984	8543	59.1	4 m, 32 n
E-7	10	2075	5201	21.9	5 m, 3 n, 2 w
E-8	10	1915	5437	20.7	4 m, 4 n, 2 w
E-9	15	2091	5700	25.1	3 m, 6 n, 6 w
E-10	23	1949	9837	42.5	7 m, 7 n, 9 w

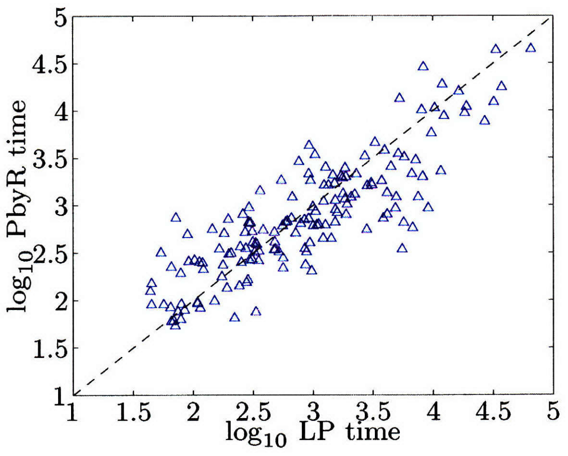


Figure 4-3: Comparison of LP times and PbyR times in pruning subproblems from branch-and-bound.

The bounding times of the two methods are comparable although LP is slightly faster in small to medium-sized subproblems. However, Figure 4-4 shows that the bounds from PbyR are greater than LP bounds except for the one data point below the $y = x$ line. Note that a PbyR bound for a subproblem is not generally a lower-bound of the subproblem's optimal value since rotamer/rotamer-pair elimination by TRMP lower-bounds can also increase the optimal value. However, a PbyR bound is greater than U only if the original subproblem's optimal value is greater than U. Therefore, if we had U equal to the GMEC energy for each design case, we could immediately prune the subproblems corresponding to the data points over the horizontal solid line in Figure 4-4. There was no such case with LP among the tested subproblems. Figure 4-4 suggests that performing reductions more than 50 times often resulted in lower-bounds that were useless for pruning.

4.6 Conclusions

In this section, we described techniques for computing lower bounds or pruning a subproblem based on problem-size reduction in the branch-and-bound framework. Rotamer contraction and edge deletion can be used to facilitate computation of lower bounds by simplifying the problem in a controlled manner. The modified dead-end

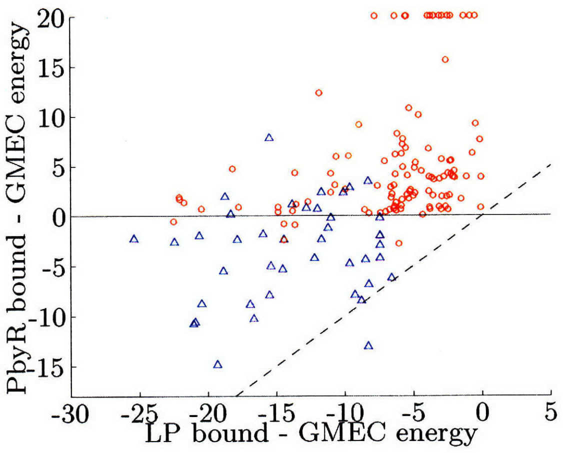


Figure 4-4: Comparison of LP bounds and PbyR bounds in pruning subproblems from branch-and-bound. A circle represents the PbyR bound was computed using less than 50 reductions. Points such that PbyR bound - GMEC energy ≥ 20 were all clamped at 20

elimination can be used with general pair flags to reduce the size of a subproblem without sacrificing any accuracy. Our computational experiments confirm that a pruning scheme combining the techniques of the current chapter and elimination by TRMP bounds of Chapter 3 can provide better pruning performance than the general-purpose linear programming solver.

Chapter 5

Computational experiments with BroMAP

We performed computational experiments to evaluate the performance of BroMAP. We used a set of various protein design cases to measure and compare the running times of BroMAP and a fast implementation of DEE/A^* that includes most of the state-of-art techniques [Gordon et al., 2003]. In the following, to distinguish the modified version of DEE used in BroMAP from the DEE used in DEE/A^*, we will call the former as *DEE-gp* (DEE for general pair flags). The two main questions we are interested in investigating with the experiments are (1) whether BroMAP can solve design cases previously unsolved by DEE/A^*, and (2) whether we can use BroMAP generally as an alternative to DEE/A^* without being restricted to specific types of design cases. We are mainly interested in the overall performance of BroMAP compared to DEE/A^* here whereas Section 4.5 evaluated the effectiveness of our pruning method by comparing it against linear programming.

5.1 DEE/A^* implementation

We used an in-house implementation of DEE/A^* written in the C programming language [Altman, 2006]. DEE/A^* was performed with the following options and order:

1. Eliminate singles using Goldstein's condition [Goldstein, 1994]. Repeat until elimination is unproductive.

2. Eliminate singles using split flags ($s = 1$) [Pierce et al., 2000]. Repeat until elimination is unproductive.

3. Do logical singles-pairs elimination [Lasters et al., 1995].

4. Eliminate pairs using Goldstein's condition with one magic bullet [Gordon and Mayo, 1998].

5. Do logical singles-pairs elimination.

6. If unification is possible, do unification [Goldstein, 1994], and go to (1).

7. Do A^* [Hart et al., 1968].

For unification, the pair of positions that has the largest fraction of flagged rotamer pairs is picked. However, because the energy terms and pair flags must be stored in machine memory, we capped the total number of rotamers that would result to be no greater than a unification option C_{uni}. Therefore, any pair of positions that would create a larger number of rotamers when unified than C_{uni} was not considered, and the pair with the next-largest fraction of flagged rotamer pairs was considered. We experimented with different values of C_{uni}, i.e. 6,000, 8,000, 10,000, 12,000, and 14,000, to obtain the best running time for each test case. Note that this gives DEE/A^*, the competing method an advantage over BroMAP in comparing their running times, because it will give better DEE/A^* times than consistently using one of the C_{uni} values. Increasing C_{uni} and thus the allowance for large unification can facilitate solving otherwise difficult or unsolvable cases. However, for small to medium cases, larger values of C_{uni} often result in slower solution times.

Our DEE implementation uses a full table of energies. That is, if there are $q = \sum_{i=1}^{n} |R_i|$ rotamers in the problem, DEE allocates memory for q^2 floating point numbers.

When the DEE/A^* procedure described above using various C_{uni} values failed to solve a test case, we also tried singles-elimination using split flags with $s = 2$ instead of $s = 1$, or allowed the number of magic bullets to increase up to the number of positions.

5.2 BroMAP implementation

BroMAP was implemented in C++. We used the PICO-library [Eckstein et al., 2001] for the BnB framework. The PICO-library provides the data structures and methods to create/delete nodes and to search the tree. It also provides procedure skeletons, for instance, for upper/lower-bounding methods.

In BroMAP, we restricted the amount of effort spent by *DEE-gp* instead of allowing it to keep iterating singles/pairs-flagging and unification until it finally solved the subproblem. This was done by limiting the maximum number of iterations of singles/pairs-flagging and also by using a smaller fixed C_{uni} value for unification than those used by DEE/A^*.

Other than performing *DEE-gp* and TRMP bounding for each subproblem, we also allowed rotamer contractions (see Section 4.2). Rotamer contraction reduces the size of a subproblem by grouping similar rotamers at a residue position as a cluster and replacing the cluster by a new single rotamer. It also defines the pairwise energies for the new rotamer so that the optimal value of the reduced subproblem is always a lower bound of the optimal value of the subproblem before the rotamer contraction. Rotamer contraction was iteratively performed until we obtained a lower bound greater than U or the number of executed rotamer contractions reached a predetermined limit. We used a heuristic boundability index (BI) multiplied by a positive integer P_{rc} as such limits. The BI for a specific node is equal to the number of 'high' branches on the path from the root to the node. For example, in the search tree of Figure 3-1, assuming the BI of the root node is equal to 0, BI's are 0 for nodes 1, 3, 5, 7, 9, and 1 for nodes 2, 4, 6, 8, 11, and 2 for node 10. Computation of BI is illustrated in Figure 5-1. In these experiments, we let $P_{rc} = 16$ after exploring the

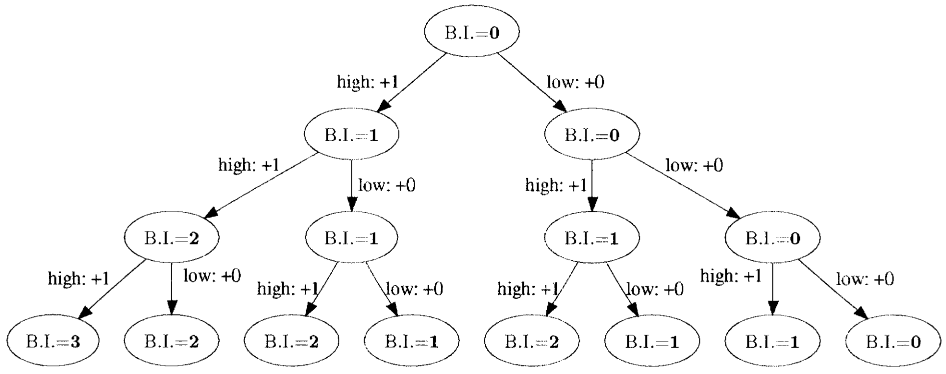


Figure 5-1: The path from root node to the current node determines the boundability index of the current node by counting the number of high branches in the path. Assuming the root node has boundability index equal to 0, the diagram shows boundability indices for all nodes in the tree.

overall effect of different values of P_{rc} on running times of BroMAP.

In case rotamer contractions were performed multiple times in bounding a subproblem as described above, we also performed additional *DEE-gp* and TRMP periodically on the subproblem reduced by rotamer contractions. After every P_{DEE} consecutive rotamer contractions, we applied *DEE-gp* to see if we could solve the reduced problem or only to flag more rotamers or rotamer pairs. TRMP was also run until convergence after every P_{TRMP} consecutive rotamer contractions to compute a lower bound for the subproblem or to flag rotamers or rotamer-pairs using the TRMP lower bounds. In this experiment, we let $P_{DEE} = 8$, and $P_{TRMP} = 16$.

Along the first depth-first dive, that is, until we exactly solve a subproblem for the first time, we performed only *DEE-gp*, TRMP bounding, and subproblem splitting, once respectively, but did not use any rotamer contraction. As with DEE/A^*, BroMAP also used the A^* search algorithm when *DEE-gp* could not eliminate any more rotamers or rotamer pairs and the subproblem was considered small, i.e. contained less than 200,000 rotamer pairs.

The BroMAP implementation needs to hold TRMP data, whose size is of the order of the number of rotamer pairs. This corresponds to $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} |R_i||R_j|$ floating point numbers, and is roughly half the memory required by DEE/A^*. Since BroMAP also performs *DEE-gp*, it requires additional memory of $(\sum_{i=1}^{n} |R_i|)^2$ floating point numbers for the full DEE energy table. Therefore, the maximum memory requirement of BroMAP is $(\sum_{i=1}^{n} |R_i|)^2 + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} |R_i||R_j|$ floating point numbers, which is roughly 1.5 times larger than that of DEE/A^*.

5.3 Platform

We used a Linux workstation with two dual-core 2 GHz AMD Opteron 246 processors and 2 Gbytes of memory for the experiment. The C/C++ codes for BroMAP and DEE/A^* were compiled using Intel C/C++ Compiler Version 9.1 for Linux. During compile, OpenMP directives were enabled to parallelize the execution of DEE, *DEE-gp*, and TRMP over two CPU cores. All other procedures, including A^*, were executed over a single core. Note that BroMAP or DEE/A^* was allowed to use the whole system memory but only one processor at a time.

5.4 Test cases

We used 68 test cases whose energy files are smaller than 300 Mbytes. All test cases were provided by Shaun M. Lippow from Tidor Group together with helpful discussions. An energy file contains floating point numbers representing singleton and pairwise energies. We found energy files larger than 300 Mbytes are not handled well with the current implementation of BroMAP on our workstation due to the memory requirement of BroMAP.

We used three different model systems in obtaining test cases:

1. FN3: derived from protein ^{10}Fn3, the tenth human fibronectin type III domain [Main et al., 1992]. It is a 94-residue β-sheet protein with an immunoglobulin-like fold. Besides its natural *in vivo* role, the protein has been engineered as an antibody mimic to bind with high affinity and specificity to arbitrary protein targets.

2. D44.1 [Braden et al., 1994] and D1.3 [Bhat et al., 1994]: antibodies that bind to hen egg-white lysozyme (HEL), though they bind different HEL epitopes. Each

has low nanomolar binding affinity, and was originally isolated after murine immunization. For the D1.3 core designs, we redesigned the core of the lysozyme protein, absent of the antibody.

3. EPO: human erythropoietin (Epo) protein complexed to its receptor (EpoR) [Syed et al., 1998]. One Epo binds to two EpoR with one high-affinity and one low-affinity binding sites. Our EPO interface designs addressed the high-affinity binding site while our core designs addressed the core of the EpoR from the high-affinity site.

Each case corresponds to one of three types of protein regions:

1. INT: protein–protein binding interface.
2. CORE: protein core, i.e. side chains that are not solvent-exposed.
3. CORE++: protein core plus boundary positions that are partially exposed to solvent.

We varied the types of amino acids offered at design positions of each case as follows:

1. H: hydrophobic amino acids (A, F, G, I, L, M, W, V).
2. HP: hydrophobic plus polar amino acids (A, F, G, I, L, M, W, V, H, N, Q, S, T, Y).
3. A: all type of amino acids, excluding proline and cysteine.

For CORE, we used both H and HP, and for CORE++, we used HP (with both neutral tautomers of histidine allowed in each case). For INT, we used A, and allowed both neutral tautomers and the protonated form of histidine. For all designs, if the wild-type amino acid was not part of the library, it was added at that position. For some test cases, the total number of positions in the search was greater than the number of design positions. At these other positions, the native amino-acid type was retained and its conformation was varied.

Each case was made using one of two different rotamer libraries:

1. REG: standard rotamer library. This is based on the backbone-independent May 2002 library [Dunbrack, 2002]. This library was supplemented with three histidine rotamers for an unsampled ring flip, and two asparagine rotamers to increase sampling of the final dihedral angle rotation.

2. EXP: expanded rotamer library. This was created by expanding both χ_1 and χ_2 of rotamers in REG by $\pm 10°$. The hydroxyls of serine, threonine, and tyrosine were sampled every 30 degrees. For some INT cases of D1.3, D44.1, and EPO, crystallographic water molecules were allowed conformational freedom. The oxygen atom location was fixed to that of the crystal structure and the hydrogen atoms were placed to create 60 symmetric water molecule rotations.

For all libraries and cases, each crystallographic wild-type rotamer was added in a position-specific manner to the library, using the complete Cartesian representation of the side chain, rather than just the dihedral angles.

The singleton/pairwise energies of rotamers were computed using the energy function of CHARMM PARAM22 all-atom parameter set with no cut-offs for non-bonded interactions and a $4r$ distance-dependent dielectric constant. All energy terms were used (bond, angle, Urey-Bradley, dihedral, improper, Lennard-Jones, and electrostatic). Rotamers that clashed with the fixed protein regions were eliminated during case generation if their singleton energies were greater than the smallest singleton energy at that position by at least 50 kcal/mol. Further details on design methods and test case construction can be found from Lippow et al. [2007]

Table 5.1: Test case facts. Each column represents (1) No.: case number, (2) Model: model system, (3) Region: protein regions being considered, (4) AA: type of amino acids offered for design positions, (5) Lib: types of rotamer library used, (6) n: number of positions, (7) n_D: number of design positions, (8) w: number of mobile water molecules considered, (9) $\sum |R_i|$: total number of rotamers, (10) Pairs: total number of rotamer pairs, (11) log*conf*: $\sum_{i=1}^{n} \log |R_i|$, (12) Solved by: methods that solved the case ("Limited DEE" implies the case was solved by both BroMAP and DEE/A^*, but only *DEE-gp* was necessary for BroMAP. "Bro" and "DEE" abbreviate BroMAP and DEE/A^*, respectively).

No.	Model	Region	AA	Lib	n	n_D	w	$\sum \|R_i\|$	Pairs	log*conf*	Solved by
1	fn3	core	HP	REG	14	14	0	743	2.5E5	50.2	Limited DEE
2	fn3	core++	HP	REG	20	20	0	1,778	1.5E6	83.7	Bro & DEE
3	fn3	core++	HP	REG	23	23	0	1,894	1.7E6	94.1	Bro & DEE
4	fn3	core++	HP	REG	25	25	0	2,048	2.0E6	102.9	Bro & DEE
5	fn3	core++	HP	REG	27	27	0	2,083	2.1E6	108.6	Bro & DEE
6	fn3	core	HP	EXP	14	14	0	8,774	3.5E7	82.4	Limited DEE
7	D44.1	int	A	REG	7	4	0	476	8.5E4	21.6	Limited DEE
8	D44.1	int	A	REG	7	7	0	822	2.8E5	28.7	Limited DEE
9	D44.1	int	A	REG	8	8	0	965	4.0E5	33.4	Bro & DEE
10	D44.1	int	A	REG	9	9	0	1,019	4.5E5	37.1	Bro & DEE
11	D44.1	int	A	REG	10	10	0	1,133	5.6E5	40.6	Bro & DEE
12	D44.1	int	A	REG	11	11	0	1,376	8.4E5	46.4	Bro
13	D44.1	int	A	REG	16	14	2	2,020	1.9E6	70.1	None

continued on next page

continued from previous page

No.	Model	Region	AA	Lib	n	n_D	w	$\sum\lvert R_i\rvert$	Pairs	log*conf*	Solved by
14	D44.1	int	A	EXP	7	4	0	5,026	9.5E6	36.4	Limited DEE
15	D44.1	int	A	EXP	7	5	0	7,019	1.9E7	39.9	Bro & DEE
16	D44.1	int	A	EXP	7	6	0	7,910	2.6E7	42.9	Bro
17	D44.1	int	A	EXP	7	7	0	8,771	3.2E7	42.9	Bro
18	D1.3	int	A	REG	6	4	2	450	8.3E4	21.7	Limited DEE
19	D1.3	int	A	REG	11	8	3	767	2.6E5	38.5	Limited DEE
20	D1.3	int	A	REG	23	7	9	1,618	1.2E6	78.8	Limited DEE
21	D1.3	int	A	EXP	6	4	2	3,599	4.8E6	28.7	Limited DEE
22	D1.3	int	A	EXP	7	5	2	3,616	4.8E6	28.7	Limited DEE
23	D1.3	int	A	EXP	8	6	2	4,070	6.3E6	34.4	Bro & DEE
24	D1.3	int	A	EXP	11	4	3	4,612	8.0E6	42.6	Bro & DEE
25	D1.3	int	A	EXP	11	6	3	4,987	9.7E6	45.1	Bro & DEE
26	D1.3	int	A	EXP	11	7	3	5,461	1.2E7	47.4	Bro & DEE
27	D1.3	int	A	EXP	11	7	3	5,891	1.4E7	50.5	Bro
28	D1.3	int	A	EXP	11	8	3	6,365	1.7E7	52.8	Bro
29	D1.3	core	H	REG	16	16	0	342	5.4E4	44.1	Limited DEE
30	D1.3	core	H	REG	20	20	0	430	8.6E4	54.6	Limited DEE
31	D1.3	core	H	REG	26	26	0	503	1.2E5	66.7	Limited DEE
32	D1.3	core	H	REG	34	34	0	567	1.5E5	81.4	Limited DEE
33	D1.3	core	HP	REG	16	16	0	980	4.4E5	59.5	Bro & DEE
34	D1.3	core	HP	REG	20	20	0	1,228	7.1E5	74.1	Bro & DEE
35	D1.3	core	HP	REG	26	26	0	1,431	9.7E5	92.3	Bro & DEE
36	D1.3	core	HP	REG	34	34	0	1,582	1.2E6	112.7	Bro & DEE
37	D1.3	core	H	EXP	13	13	0	1,844	1.5E6	56.3	Bro & DEE
38	D1.3	core	H	EXP	16	16	0	2,734	3.5E6	75.7	Bro
39	D1.3	core	H	EXP	20	20	0	3,370	5.3E6	91.8	Bro

continued on next page

continued from previous page

No.	Model	Region	AA	Lib	n	n_D	w	$\sum\lvert R_i\rvert$	Pairs	log*conf*	Solved by
40	D1.3	core	H	EXP	26	26	0	3,894	7.1E6	111.6	Bro
41	D1.3	core	H	EXP	34	34	0	4,444	9.4E6	142.0	Bro
42	epo	int	A	REG	5	5	0	466	7.1E4	16.6	Limited DEE
43	epo	int	A	REG	6	6	0	419	6.8E4	17.0	Limited DEE
44	epo	int	A	REG	11	11	0	1,005	4.4E5	39.4	Bro & DEE
45	epo	int	A	REG	21	11	3	1,503	1.0E6	67.5	Bro & DEE
46	epo	int	A	REG	21	15	3	1,999	1.9E6	79.6	Bro
47	epo	int	A	REG	21	18	3	2,138	2.1E6	87.5	None
48	epo	int	A	EXP	5	5	0	5,001	8.4E6	26.5	Limited DEE
49	epo	int	A	EXP	6	6	0	4,170	6.8E6	26.3	Bro & DEE
50	epo	int	A	EXP	8	8	0	7,544	2.3E7	46.4	Bro & DEE
51	epo	int	A	EXP	9	9	0	8,724	3.2E7	53.4	Bro & DEE
52	epo	core	H	REG	17	17	0	291	3.9E4	43.5	Limited DEE
53	epo	core	H	REG	22	22	0	395	7.4E4	58.1	Limited DEE
54	epo	core	H	REG	28	28	0	433	8.9E4	65.4	Limited DEE
55	epo	core	H	REG	33	33	0	573	1.6E5	82.7	Limited DEE
56	epo	core	H	REG	41	41	0	727	2.6E5	103.3	Limited DEE
57	epo	core	HP	REG	17	17	0	827	3.2E5	60.1	Bro & DEE
58	epo	core	HP	REG	22	22	0	1,103	5.8E5	79.9	Bro & DEE
59	epo	core	HP	REG	28	28	0	1,208	7.0E5	92.6	Bro & DEE
60	epo	core	HP	REG	33	33	0	1,615	1.3E6	115.9	Bro & DEE
61	epo	core	HP	REG	36	36	0	1,827	1.6E6	128.4	Bro
62	epo	core	HP	REG	38	38	0	1,956	1.9E6	136.6	Bro
63	epo	core	HP	REG	41	41	0	1,999	1.9E6	143.1	Bro
64	epo	core	H	EXP	17	17	0	2,307	2.4E6	73.5	Limited DEE
65	epo	core	H	EXP	22	22	0	3,006	4.2E6	99.0	Bro & DEE

continued on next page

continued from previous page

No.	Model	Region	AA	Lib	n	n_D	w	$\sum\|R_i\|$	Pairs	log*conf*	Solved by
66	epo	core	H	EXP	28	28	0	3,213	4.8E6	111.1	Bro & DEE
67	epo	core	H	EXP	33	33	0	4,322	8.9E6	140.0	Bro
68	epo	core	H	EXP	41	41	0	5,712	1.6E7	175.0	None

Table I lists composition and problem-size information of each test case. Its last column also summarizes the experimental results presented in the following.

5.5 Running time comparison

Among the 68 cases, BroMAP solved 65 cases within the 7-days allowed time whereas DEE/A^* solved 51 cases for the same allowed time. There were no cases DEE/A^* solved but BroMAP was not able to solve. The 14 cases solved by BroMAP but not by DEE/A^* suggest that BroMAP can be an alternative to DEE/A^* for hard design cases where DEE/A^* performs poorly.

Among the 51 cases solved by both BroMAP and DEE/A^*, solving 23 cases by BroMAP required only the *DEE-gp* part of BroMAP. Since BroMAP only acted as a light DEE for these cases, comparing the running times of BroMAP and DEE/A^* on them is not meaningful. After eliminating these 23 cases, we are left with 28 cases for which we are interested in comparing the running times of BroMAP and DEE/A^*. The running times for these 28 cases are shown in Table II. Additionally, the table lists results for 14 cases that only BroMAP solved.

Table 5.2: Results of solving the non-"Limited DEE" cases with BroMAP and DEE/A^* (cases solved by limited DEE are not presented). Columns (1) No.: case number, (2) Bro: BroMAP solution time in seconds, (3) DEE: DEE/A^* solution time in seconds, (4) T-Br: total number of branchings (i.e. splits), (5) F-Br: number of branchings during the first depth-first dive, (6) Skew: skewness of the search tree defined as $\frac{\text{(number of low-subproblems split)}}{\text{(total number of splits) - 1}}$, (7) F-Ub: $U - OPT$, i.e. difference between the upper bound from the first depth-first dive and the GMEC energy, (8) Leaf: $\sum_i \log_{10}|R_i|$ of the node at the end of the first depth-first dive, (9) Rdctn: average reduction of $\sum_i \log_{10}|R_i|$ during the first depth-first dive, i.e. (log*conf* - Leaf)/(F-Br), where log*conf* is defined in Table I, (10) RC: number of rotamer contractions performed, (11) %DE: BroMAP time percentage used for *DEE-gp*, (12) %A^*: BroMAP time percentage used for A^*, (13) %TR: BroMAP time percentage used for TRMP. Note that columns 11 to 13 may not sum to 100% because of time spent on rotamer contraction and overhead of using the branch-and-bound framework.

No.	Bro	DEE	T-Br	F-Br	Skew	F-Ub	Leaf	Rdctn	RC	%DE	%A^*	%TR
2	2.6E3	3.1E4	31	25	0.90	0.49	30.7	2.12	36	42.8	0.3	56.3
3	2.4E3	2.3E4	31	26	0.93	0.49	27.7	2.55	32	46.2	0.6	52.6
4	2.8E3	1.3E4	23	23	1	0	33.7	3.01	0	43.9	0.3	55.5
5	2.7E3	2.1E4	26	26	1	0.55	27.4	3.12	0	37.2	0.4	62.2
9	1.2E2	4.8E2	3	3	1	0	27.6	1.93	0	8.9	74.1	17.0

continued on next page

continued from previous page

No.	Bro	DEE	T-Br	F-Br	Skew	F-Ub	Leaf	Rdctn	RC	%DE	%A^*	%TR
10	4.6E2	1.3E3	13	10	0.75	0.37	26.9	1.02	74	7.6	70.4	14.4
11	5.7E3	3.5E4	109	17	0.81	0.36	26.2	0.85	663	3.8	78.9	11.2
15	2.9E2	3.5E2	0	0	NA	0	NA	NA	0	94.6	0.4	4.7
23	1.5E2	2.6E2	0	0	NA	0	NA	NA	0	86.7	0	12.6
24	3.2E2	3.1E2	4	4	1	0	25.3	4.33	0	62.3	15.1	21.6
25	2.9E2	1.2E3	0	0	NA	0	NA	NA	0	89.6	0	10.4
26	1.4E3	1.7E3	11	11	1	0.89	29.2	1.65	0	46.1	0.4	53.2
33	4.1E2	2.1E3	13	13	1	0	27.9	2.43	0	34.7	4.5	59.8
34	1.1E3	3.7E3	19	19	1	0	30.0	2.32	0	32.2	2.7	64.8
35	2.8E3	4.1E4	21	21	1	0	28.7	3.03	0	50.7	0.6	48.6
36	4.6E3	2.3E4	25	25	1	0	27.9	3.39	0	53.2	0.7	45.9
37	2.5E2	2.5E2	0	0	NA	0	NA	NA	0	76.0	2.4	21.2
44	2.2E2	3.8E1	8	6	0.71	0.54	28.2	1.87	17	8.2	75.5	14.1
45	8.8E2	2.0E2	8	8	1	0	26.2	5.16	8	48.6	23.8	25.4
49	3.3E2	5.0E2	4	4	1	0	19.8	1.63	0	51.1	11.5	37.5
50	1.2E3	1.1E3	7	7	1	0	22.3	3.44	12	72.2	7.0	17.1
51	5.7E4	2.8E5	666	25	0.85	0.58	27.6	1.03	5,656	16.7	21.2	41.8
57	4.6E1	2.7E2	0	0	NA	0	NA	NA	0	84.8	0	15.2
58	1.5E3	1.0E3	19	19	1	0	28.8	2.69	0	42.5	0.2	57.1
59	4.4E2	4.0E3	0	0	NA	0	NA	NA	0	70.6	0	29.1
60	1.5E4	4.6E4	32	32	1	0	37.3	2.46	0	30.1	0.1	69.7
65	4.6E3	1.7E3	15	15	1	0	22.7	5.09	0	61.9	0	37.8
66	7.7E3	2.4E3	15	15	1	0	33.9	5.15	0	67.2	0	32.6
												Cases below were solved by BroMAP only.
12	2.0E5	NA	2773	23	0.82	7.11	26.2	0.88	3.9E4	6.0	59.1	20.1
16	3.5E3	NA	12	11	0.91	0	23.6	1.75	30	41.7	6.0	49.3

continued on next page

continued from previous page

No.	Bro	DEE	T-Br	F-Br	Skew	F-Ub	Leaf	Rdctn	RC	%DE	%A^*	%TR
17	1.1E5	NA	298	21	0.84	3.35	26.7	0.77	2,576	17.7	28.1	32.7
27	8.0E3	NA	23	23	1	0	27.8	0.99	13	32.2	1.1	66.4
28	2.1E4	NA	175	25	0.91	0	28.0	0.99	1,168	23.8	8.5	57.9
38	1.4E4	NA	155	31	0.87	0.50	30.2	1.47	571	30.9	0.4	62.8
39	1.2E5	NA	572	43	0.85	0	27.4	1.50	4,791	30.4	0.1	58.4
40	1.8E5	NA	293	43	0.81	0	29.6	1.91	2,440	35.9	0	56.1
41	2.1E5	NA	364	41	0.85	0	33	2.66	2,771	34.2	0	57.5
46	5.0E5	NA	2675	36	0.69	8.28	27.8	1.44	1.4E5	18.8	18.8	35.3
61	2.8E4	NA	55	49	0.96	0.36	28.2	2.04	15	49.0	0	50.8
62	3.6E5	NA	232	58	0.88	0.27	30.1	1.84	1,119	43.5	0	50.2
63	1.1E5	NA	143	53	0.85	0.29	32.8	2.08	506	41.4	0	55.4
67	1.3E5	NA	37	37	1	0	35.6	2.82	0	51.5	0	48.5

Figure 5-2 plots the ratio of BroMAP running time to DEE/A^* running time vs. DEE/A^* running time. Note that the plotted ratios for cases solved only by BroMAP are upper bounds on actual ratios because actual DEE/A^* running times should be more than 7 days. Overall, the plot suggests BroMAP gains advantage for cases as DEE/A^* takes longer. For all cases that DEE/A^* took more than one hour to solve, the maximum ratio was 0.33. Together with the 14 cases solved by BroMAP only, the experiment supports that BroMAP can be an alternative to DEE/A^* for hard design cases. There are 5 cases for which the BroMAP solution time is at least 10% longer than DEE/A^* solution time. Considering four of them (cases 45, 58, 65, and 66) were almost ideally solved by BroMAP (the GMEC was found at the end of the first depth-first dive and there was no branching after the first depth-first dive), we find more aggressive DEE conditions such as larger C_{uni} were critical in obtaining shorter running times on them. In terms of the total running time, however, none of these five cases needed more than 130 minutes to be solved by BroMAP. Therefore, using BroMAP did not impractically slow down the solution time for cases in Table

I.

For large hard cases, the system memory can be a limiting factor on the performance of DEE/A^* because the performance of DEE/A^* often greatly depends on the unification procedure that requires a large amount of memory. While this implies larger system memory could have given advantage to DEE/A^* over BroMAP in terms of running time, our results suggests that the memory constraints experienced by DEE/A^* can be alleviated through the use of BroMAP.

Table II lists the percentage of time used for each component of BroMAP. In most cases, *DEE-gp*, A^*, and TRMP turned out to be major contributors to the running time. If we sum running times of BroMAP for all cases, it is found that 42% of the total time was spent on *DEE-gp* and A^*, and 45% on TRMP. On the other hand, considering the proportion between the total running time of BroMAP and A^* time, a great amount of time was spent on A^* for cases 11 and 12. This could be avoided by further restricting the size of the subproblem for which A^* is allowed to run.

Among cases in Table I, BroMAP was able to solve six cases at the root node without splitting. All other cases required BroMAP to branch but many of them needed very little branching other than those performed during the first depth-first dive. This trend is observed through the skewness of the search tree, defined as $\frac{\text{(number of low-subproblems split)}}{\text{(total number of branchings) - 1}}$. The ratio varies between 0 and 1 and is larger than 0.5 if there are more low-subproblems split than high-subproblems. We computed skewness for 36 cases where BroMAP required more than one split. The minimum skewness from these cases is 0.69 and 17 cases had skewness equal to 1, that is, needed only low-subproblem splittings.

Figure 5-3 shows actual search trees generated by BroMAP during solution of three cases. Box-shaped (shaded) nodes in each search tree represent the subproblems that were exactly solved and resulted in an upper bound less than equal to the current best upper bound. Therefore, the box-shaped node that is expanded latest is a node where the GMEC is found in the search tree. Note that, for 27 cases out of 42 cases in Table II, an upper bound equal to the GMEC energy was found at the end of the first depth-first dive. However, early discovery of the GMEC did not necessarily lead

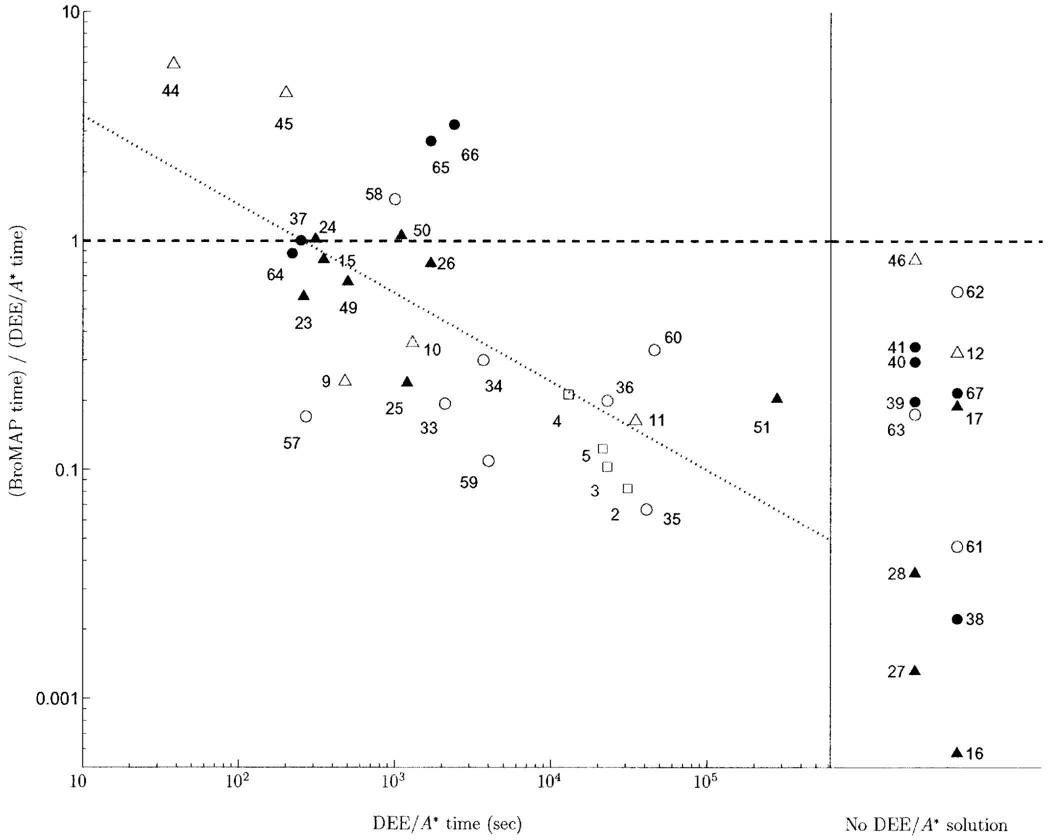
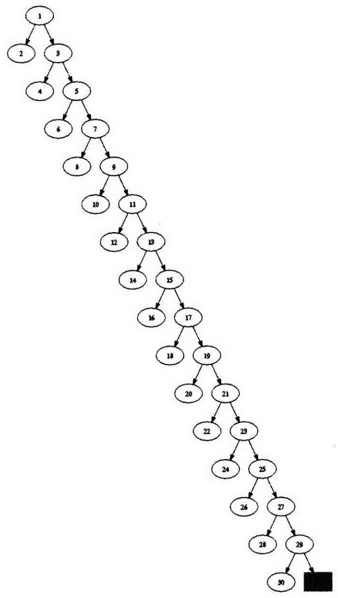


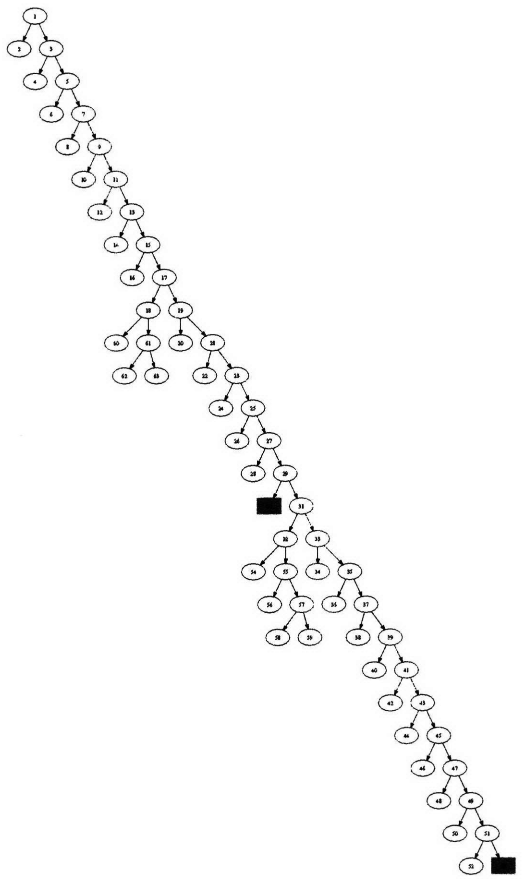
Figure 5-2: Ratio of BroMAP time to DEE/A^* time vs. DEE/A^* time for 42 cases in Table II. Labels next to data points are case numbers from Table I. The 14 cases solved by BroMAP only are shown in the narrow pane on the right side. The running time ratios for these cases were calculated by assuming the DEE/A^* time for each of them is 7 days although they were not solved by DEE/A^* within 7 days. The trend line represents a robust fit for the 28 cases that were solved by both BroMAP and DEE/A^*. The horizontal dashed line represents the ratio equal to 1. Different symbols are used to represent each case depending on the type of protein region (CORE, CORE++, or INT) and the type of library used (REG or EXP): (1) ○ = CORE, □ = CORE++, Δ = INT, (2) empty = REG, filled = EXP.

to fast completion of BroMAP. For example, in Figure 5-3(c), we can see the lower bounding was not effective for large subproblems although they were expanded after the optimal upper-bound was found, resulting in a large number of branchings.

Table II suggests that the search trees of BroMAP have smaller depths than those from conventional BnB methods would have. A simple branching without reduction within a node would only reduce the problem size by a factor of two. That is, a child subproblem will have $\sum_i \log_{10} |R_i|$ value smaller by $\log_{10} 2 \approx 0.30$ than its parent subproblem. However, column "Rdctn" shows the average reduction was far greater. Excluding the cases solved without branching, the average of the average reduction of $\sum_i \log_{10} |R_i|$ along the first depth-first dive was 2.32, a factor of 7 speed-up over reduction by conventional BnB methods. It should be noted that the reduction within a node can be even greater after a strong upper bound is found. This is evidenced by highly skewed shapes of search trees. Overall, the reduced depth and high skewness of BroMAP search trees suggest the number of nodes expanded by BroMAP is exponentially smaller than that of conventional BnB methods using simple branching. Meanwhile, the effect of smaller search trees will be transferred to shorter running times as well; the experimental results presented in Section 4.5 show that the node processing time by *DEE-gp* and TRMP is similar to the bounding time for solving a linear programming (LP) problem, a typical bounding method used in BnB methods, but the LP produces weaker bounds.

The plots in Figure 5-4 provide interesting insights on the hardness of test cases. In Figure 5-4(a), categorizing all test cases by their solvability reveals cases with higher ratios of log conformation to the number of design positions tend to be harder to solve. Figure 5-4(b) uses gray scale to represent the running times of BroMAP. Although the performance of BroMAP is not particularly dependent on protein regions, it is noted that INT cases are smaller than CORE cases. This is because we excluded small CORE cases from the experiment because they are often too easy for either BroMAP or DEE/A^*, and also excluded large INT cases for the opposite reason. There are two main reasons that INT cases are harder than CORE. First, INT cases are offered more rotamers per position because we allowed 8 to 14 amino acids for CORE cases


(a) Case 65 (skew = 1.0)


(b) Case 3 (skew = 0.93)

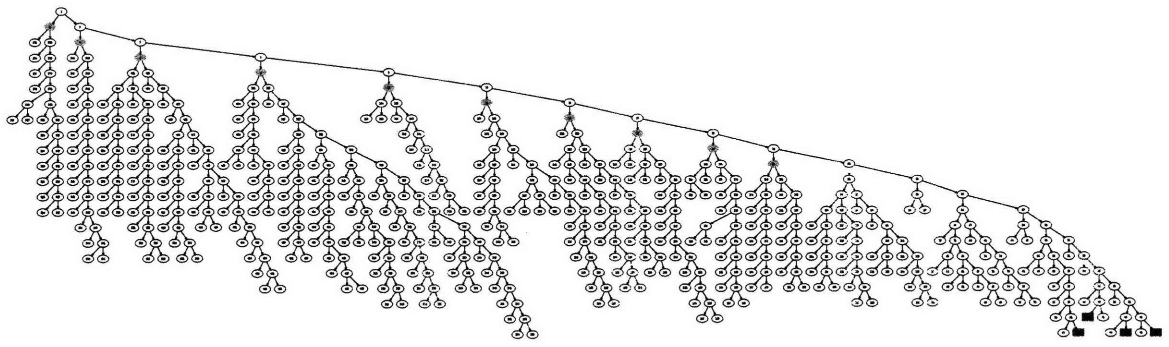

(c) Case 17 (skew = 0.84)

Figure 5-3: Search trees of BroMAP for three cases. For each branching, the low-subproblem is placed on the right-hand side, and the high-subproblem on the left-hand side. Shaded box-shaped nodes represent the subproblems that were exactly solved and resulted in an update of the global upper bound.

whereas 18 amino acids including R, K, D and E were allowed for INT cases. These four additional amino acids offer even more rotamers per amino acid than average because of their long side chains. Second, whereas CORE cases are constrained by side-chain/side-chain interactions as well as side-chain/backbone interactions, INT cases are generally less constrained by side-chain/backbone interactions, and therefore there exist a larger number of compatible conformations.

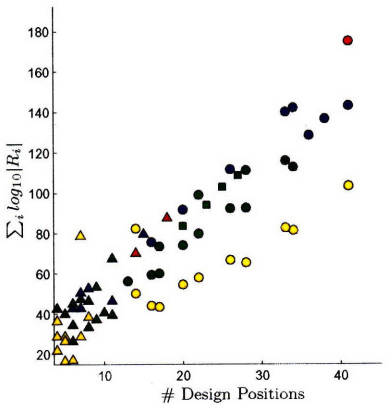
5.6 TRMP lower bounds

We present a numerical example to illustrate the utility of TRMP lower bounds in rotamer/rotamer-pair elimination. For this purpose, we use subproblems of Case 17 at depth 2 to 11 (root node is at depth 1). These subproblems correspond to node numbers $2, 4, 6, \ldots, 20$, and are colored in light gray in Figure 5-3(c) (nodes in the search tree are numbered by the order of creation using depth-first search). Table III lists the lower-bounding result.

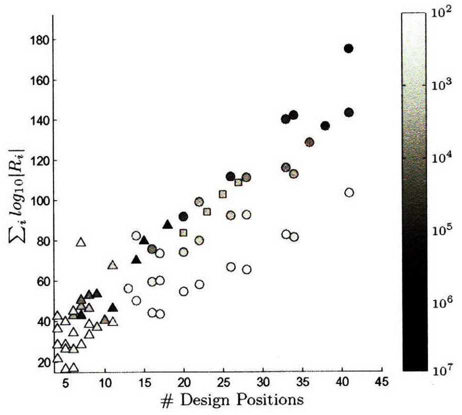
In each node, we obtain more elimination using $rotlb_2$ (rotamer lower bound from using pair flags) than using $rotlb_1$ (rotamer lower bound from not using pair flags). This is due to massive flagging of rotamer pairs by *rplb* (rotamer-pair lower bound). Figure 5-5 shows rotamers ordered by the value of $rotlb_1$ on x-axis and their $rotlb_1$, $rotlb_2$ values on y-axis for the subproblem of node 2. The difference between $rotlb_1$ and $rotlb_2$ for the same rotamer shows pair-flags information can strengthen the lower bounds, thereby doubling the number of eliminated rotamers in this example.

Large elimination obtained for subproblems at small depth are suspected to come from our splitting scheme of dividing rotamers by their lower bounds. As we go deeper down the search tree, we expect such distinction between rotamer lower bounds to become less clear. The trend is observed by the median value of $rotlb_1$ and the percentage of eliminated rotamers and rotamer pairs for nodes at different depths.

Computing $rotlb_2$ takes more time than $rotlb_1$, but Table III shows that the difference is relatively insignificant compared to the time for computing *rplb*. The time for computing *rplb* for every rotamer pair was typically at least double the time for TRMP



(a) Solvability



(b) Protein region and BroMAP running time

Figure 5-4: Each case is plotted by the number of design positions and log number of conformations. In both (a) and (b), different symbols were used for different protein regions: (1) Δ = INT, (2) $\bigcirc$ = CORE, (3) $\square$ = CORE++. In (a), cases were marked with different colors depending on their solvability: (1) yellow = solved by limited DEE, (2) green = solved by BroMAP and DEE/A^*, (3) blue = solved by BroMAP only, (4) red = solved by none. In (b), the BroMAP running time on each case was used to color the corresponding symbol. The color bar on the right side shows mapping between a color and a running time in seconds.

Table 5.3: TRMP lower-bounding results for subproblems of Case 17. The meaning of each column is, in order: (1) node number, (2) number of rotamers in the subproblem, (3) number of rotamer pairs in the subproblem, (4) time (sec) for TRMP convergence, (5) median rotamer lower bound when not using pair flags ($rotlb_1$), (5) percentage of rotamers such that $rotlb_1 > U$, (6) time (sec) for computing $rotlb_1$ for all rotamers, (7) median rotamer-pair lower bound ($rplb$), (8) percentage of rotamer pairs such that $rplb > U$, (9) time (sec) for computing $rplb$ for all rotamer pairs, (10) median rotamer lower bound when using pair flags ($rotlb_2$) after rotamer pairs were flagged by $rplb$, (11) percentage of rotamers such that $rotlb_2 > U$, (12) time (sec) for computing $rotlb_2$ for all rotamers. In the Table, time for TRMP convergence was excluded from time for computing $rotlb_1$, $rotlb_2$ or $rplb$. The value of U is -55.13, which is equal to the optimal value and was available as a global upper bound for each node in the table by the time they were expanded.

				Rot lb's w/o p-flags			Rot-pair lb's			Rot lb's w/ p-flags		
Node	#rots	#pairs	T_{TR}	med	%el	time	med	%el	time	med	%el	time
2	4504	8.4E6	148	-71.69	11	0	-40.21	74	1162	-70.91	26	7
4	3837	6.3E6	203	-70.98	9	0	-46.18	68	774	-70.61	21	3
6	3570	5.4E6	238	-80.98	3	0	-53.45	54	607	-80.56	7	4
8	3269	4.5E6	190	-84.57	1	1	-58.82	44	463	-84.35	3	1
10	2969	3.7E6	99	-73.98	12	0	-47.80	62	353	-73.59	17	2
12	2704	3.1E6	105	-84.00	7	0	-59.17	45	261	-83.81	8	1
14	2504	2.6E6	77	-84.10	13	0	-51.16	50	202	-83.93	14	1
16	2173	2.0E6	65	-82.25	5	0	-61.36	42	138	-82.13	8	0
18	1878	1.5E6	71	-88.56	5	0	-66.31	40	92	-88.47	9	0
20	1725	1.3E6	16	-86.09	7	0	-65.68	38	74	-85.94	8	0

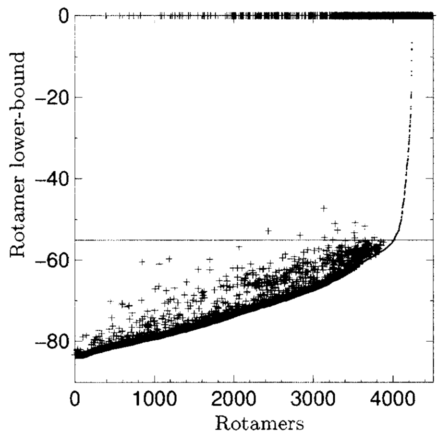


Figure 5-5: Plots of TRMP lower bound vs. rotamer, for all (4,504) rotamers in the subproblem of node 2 in solving Case 17 by BroMAP; $rotlb_1$ is represented by a dot and $rotlb_2$ by a '+' symbol. Rotamers are sorted on x-axis by the increasing order of $rotlb_1$. All rotamers with lower bounds greater than equal to 0 were clipped at $y = 0$. The horizontal line at $y = -55.13$ represents U. By comparing $rotlb_1$ against U, 497 rotamers ($4,008^{th}$ to $4,504^{th}$ in the order) were eliminated. Using $rotlb_2$ instead increased the number of eliminated rotamers to 1,171.

convergence, suggesting that an efficiency improvement of rotamer-pair lower-bound computation would significantly contribute to reducing the running time of BroMAP.

5.7 Conclusions

We performed computational experiments to evaluate BroMAP on various types and sizes of protein design problems in comparison with DEE/A^*. The experimental results show that BroMAP solved hard protein design cases faster than DEE/A^*, and that BroMAP also solved many cases that DEE/A^* failed to solve within allowed time and memory. In addition, using BroMAP on cases where DEE/A^* performed well did not incur significant disadvantage in running time.

The performance advantage of BroMAP over DEE/A^* or conventional BnB methods can be attributed to three factors. First, the search trees are radically smaller than those from conventional BnB methods. Problem-size reduction performed within each node results in reduced depths of search trees, and early discovery of suboptimal upper bounds allows effective pruning of nodes that follow. Second, on top of fast reduction by DEE within each node, BroMAP can perform additional elimination and informed branching using lower bounds from inexpensive computation. Third, the general BnB framework of BroMAP allows additional lower-bounding techniques such as rotamer contraction to be easily incorporated.

It could be argued that the performance comparison between BroMAP and DEE/A^* was not thorough or fair because DEE can be faster depending on what elimination conditions are used, how they are combined [Gordon et al., 2003], or how much memory is available for unification. However, it should be noted that BroMAP also exploits DEE, and that BroMAP can be regarded as an added structure to DEE/A^* to allow a more effective use of it in a general framework. As a result, if a better implementation of DEE/A^* is given or a better system environment is allowed, the performance of BroMAP is also expected to benefit from it.

In our experiment, using rotamer contraction did not always improve the total running time of BroMAP, although it tends to reduce the number of nodes expanded by BroMAP. However, among the 14 test cases that were solved by BroMAP with rotamer contraction but not by DEE/A^*, two could not be solved by BroMAP without rotamer contraction within the 7-day time limit. In addition, for the 51 test cases used for comparison of BroMAP and DEE/A^*, the total running time of BroMAP was reduced by 17% on average simply by using rotamer contraction. Therefore, there is a question of how much effort should be spent on rotamer contraction to maximize the performance of BroMAP. On the other hand, observing the behavior of BroMAP on many random instances to parameterize its solution time by problem characteristics will be interesting and may help improve the performance of BroMAP, because no direct correlation between the problem size and the BroMAP solution time has been found. Finally, a substantial fraction of BroMAP's running time is spent on post-processing of TRMP to compute rotamer-pair lower bounds. Therefore, a speed-up of BroMAP could be made through efficiency improvement of this post-processing procedure. Our future investigation will address these problems to extend the applicability of BroMAP to larger protein design cases.

Chapter 6

Lagrangian dual relaxation of the GMEC problem

6.1 Introduction

In Chapter 3, we have seen that the minimum conformational energy can be lower-bounded by TRMP, a message passing algorithm that finds the maximum value of the convex combination of tree distributions. It is known that the maximum value of the convex combination is equivalent to the optimal value of the linear programming relaxation over the local marginal polytope [Wainwright et al., 2005]. Such a linear program is derived as a dual problem of the original formulation of minimizing the convex combination over all tree distributions. Despite the simple appearance of the linear program over the local marginal polytope, solving the linear program for protein design problems is often very challenging. TRMP provides better scalability than the linear program solver, but the bound it finds is often weaker than the linear program bound.

In this chapter, the minimum energy lower bounding problem (or MAP estimation problem) is approached from the Lagrangian dual optimization perspective, which is on the other hand regarded as the primal problem by Wainwright et al. [2005]. Using a dual optimization framework has several advantages. Firstly, dual optimization can be more scalable than the linear program because it uses a relatively small number of Lagrangian multipliers and the subproblem can be handled by problem-specific solvers. Secondly, dual optimization is an anytime algorithm; we can stop the dual optimization iteration at any time and still obtain a lower bound. Thirdly, while TRMP sometimes has a problem with numerical convergence and has no theoretical guarantee for finding an optimal solution to the relaxation, dual methods come with theoretical results regarding convergence to the dual optimal solution. Our experiments also confirm that the dual methods find tighter bounds through the use of proper stepsize rules than TRMP or generalized MPLP [Globerson and Jaakkola, 2007a], a message-passing equivalent to the dual methods, when the same decomposition of the graphical model is used. This suggests that the message passing algorithms may experience more numerical difficulties in finding the optimal solution they intend to find than their counterpart dual methods such as the subgradient method.

Lastly, various decomposition of the problem can be easily adopted in the dual framework [Komodakis et al., 2007]. This will allow stronger bounds depending on the type of decomposition. TRMP uses tree decomposition, and its linear program equivalent is readily represented by the local marginal polytope. However, both TRMP and the linear program become cumbersome when the original distribution is decomposed into graphical models that contain cycles or clusters. In the Lagrangian dual optimization framework, the complication from using cyclic graphs is passed on to the solver of each decomposed problem.

In the following sections, we will first introduce the Lagrangian dual formulation of the MAP estimation problem. Then, we will discuss the effect of using cyclic graphs for decomposition in terms of strength of lower bounds from the Lagrangian dual problem. After that, we will describe the solution procedure using the subgradient method and the junction tree algorithm. In our computational experiments, We will explore different ways of choosing a set of graphical models for decomposition to obtain strong bounds from the subgradient method.

The formulation of Lagrangian dual relaxation is based on Komodakis et al. [2007], but we do not constrain the subgraphs used for decomposition to trees. We mathematically show that the use of cyclic subgraphs or addition of variable cliques in decomposition may lead to improvement of lower bounds using similar proof techniques used by Wainwright et al. [2005]. We also suggest a scheme for selecting triplets relevant to improving lower bounds, and incremental addition of triplets on the fly in the subgradient method.

6.2 Lagrangian relaxation

We will follow the formulation and notation presented by Komodakis et al. [2007] in describing the Lagrangian relaxation of the GMEC problem. The key idea of the Lagrangian relaxation we will see is introducing Lagrange multipliers to relax a set of constraints so that the relaxed problem reveals a separable structure that facilitates the solution of individual decomposed problems.

We start from the integer linear program (B.1)–(B.6) as our primal problem, where the energy function is represented as

$$E(x) = \sum_{i\in V}\sum_{r\in R_i} E(i_r)x_{i_r} + \sum_{(i,j)\in E}\sum_{(r,s)\in R_i\times R_j} E(i_r j_s)x_{i_r j_s}. \tag{6.1}$$

Note that x_{i_r} and $x_{i_r j_s}$ are binary variables defined for each rotamer and rotamer pair of the problem, and x is a binary vector consisting of those and has the length of $d(\mathcal{G}) \overset{def}{=} \sum_{i\in\mathcal{V}(\mathcal{G})}|R_i| + \sum_{(i,j)\in\mathcal{E}(\mathcal{G})}|R_i\times R_j|$. We denote the set of binary vectors that correspond to feasible assignments as $\mathcal{F}$. This is equivalent to the set of binary vectors satisfying (B.2)–(B.3).

To form a separable structure in the primal problem, the graphical model $\mathcal{G} = (\mathcal{V},\mathcal{E})$ is decomposed into a set of subgraphs $\{\mathcal{G}^h = (\mathcal{V}^h,\mathcal{E}^h)\}$ such that $\bigcup_h \mathcal{G}^h = \mathcal{G}$, as done in TRMP. Then, we can define a function $E^h : \{0,1\}^{d(\mathcal{G}^h)} \mapsto \mathbb{R}$ for each h such that

$$E^h(x^h) = \sum_{i\in\mathcal{V}^h}\sum_{r\in R_i} E^h(i_r)x^h_{i_r} + \sum_{(i,j)\in\mathcal{E}^h}\sum_{(r,s)\in R_i\times R_j} E^h(i_r j_s)x^h_{i_r j_s}, \tag{6.2}$$

where $x^h_{i_r} \in \{0,1\}$ and $E^h(i_r) \in \mathbb{R}$ for each $r \in R_i$, $i \in \mathcal{V}^h$ and $x^h_{i_r j_s} \in \{0,1\}$, and

$E^h(i_r j_s) \in \mathbb{R}$ for each $(r, s) \in R_i \times R_j$, $(i, j) \in \mathcal{E}^h$.

Then, the original energy function can be represented as the sum of E^h over all h:

$$E(x) = \sum_h E^h(x_{|G^h}) \text{ for all } x \in \mathbb{R}^{d(\mathcal{G})} \tag{6.3}$$

where $x_{|G^h}$ denotes a subset of x corresponding to vertices and edges in $\mathcal{G}^h$. We also define $\mathcal{F}^h$ as the set of all feasible binary vectors that correspond to feasible assignments with respect to $\mathcal{G}^h$. Then, (B.1)–(B.6) can be rewritten by introducing redundant variables x^h for each h and using $E^h(x^h)$ as follows:

$$\text{minimize} \qquad \textstyle\sum_h E^h(x^h) \tag{6.4}$$

$$\text{subject to} \quad x \in \mathbb{R}^d,\ x^h \in \mathcal{F}^h,\ x^h = x_{|G^h}\ \forall h, \tag{6.5}$$

where (6.5) enforces agreement between the value of x^h for each h and that of x so that exactly one rotamer is chosen for each position.

To exploit the separable structure, we introduce Lagrange multipliers $\lambda^h \in \mathbb{R}^{d(\mathcal{G}^h)}$ for constraints (6.5) as follows:

$$q(\{\lambda^h\}) = \min_{\{x^h \in \mathcal{F}^h\}, x} L(\{x^h\}, x, \{\lambda^h\}) = \min_{\{x^h \in \mathcal{F}^h\}, x} \sum_h E^h(x^h) + \sum_h \lambda^h \cdot (x^h - x_{|G^h}). \tag{6.6}$$

By the visualization lemma [Bertsekas, 1999], we know that $q(\{\lambda^h\})$ is a lower bound of the optimal value of (6.4)–(6.5) for all $\lambda^h \in \mathbb{R}^{d(\mathcal{G}^h)}$. The consequent weak duality theorem states the optimal value of

$$\begin{aligned} &\text{maximize} && q(\{\lambda^h\}) \\ &\text{subject to} && \{\lambda^h\} \in \mathbb{R}^{\sum_h d(\mathcal{G}^h)} \end{aligned} \tag{6.7}$$

is also a lower bound of our original problem. We call (6.7) the dual problem. In the following, we let $\tilde{d} = \sum_h d(\mathcal{G}^h)$, and $\lambda = \{\lambda^h\} \in \mathbb{R}^{\tilde{d}}$ for notational simplicity.

As intended by the formulation, our Lagrangian function $L(\{x^h\}, x, \lambda)$ has a separable structure, that is,

$$\min_{\{x^h \in \mathcal{F}^h\}, x} L(\{x^h\}, x, \lambda) = \sum_h \min_{x^h \in \mathcal{F}^h} \{E^h(x^h) + \lambda^h \cdot x^h\} - \min_x \sum_h \lambda^h \cdot x_{|G^h}. \tag{6.8}$$

Therefore, for given λ, finding $q(\lambda)$ can be significantly easier than solving the primal problem because the first minimization on the right-hand side can be solved separately for each decomposed problem. On the other hand, the term minimized over all x can be rearranged as follows

$$\sum_h \lambda^h \cdot x_{|G^h} = \sum_{i \in \mathcal{V}} \sum_{r \in R_i} x_{i_r} \sum_{h: i \in \mathcal{V}^h} \lambda^h_{i_r} + \sum_{(i,j) \in \mathcal{E}} \sum_{(r,s) \in R_i \times R_j} x_{i_r j_s} \sum_{h:(i,j) \in \mathcal{E}^h} \lambda^h_{i_r j_s}. \tag{6.9}$$

Since x is unconstrained, if $\sum_{h: i \in \mathcal{V}^h} \lambda^h_{i_r}$ is non-zero for some i_r, the term $\sum_h \lambda^h \cdot x_{|G^h}$ can be made as large as we please by tending x_{i_r} to positive or negative infinity depending on the sign of $\sum_h \lambda^h \cdot x_{|G^h}$, thereby $q(\lambda)$ to negative infinity. A similar argument can be made for a pair of rotamers as well. Since we are interested in finding the maximum value of $q(\lambda)$, we only need to solve the dual problem over $D \overset{def}{=} \{\lambda \mid q(\lambda) > -\infty\}$, that is, by the argument above,

$$\begin{aligned} D = \{\lambda \mid & \sum_{h: i \in \mathcal{V}^h} \lambda^h_{i_r} = 0, \ \forall i \in \mathcal{V}, \ r \in R_i, \\ & \sum_{h:(i,j) \in \mathcal{E}^h} \lambda^h_{i_r j_s} = 0, \ \forall (i,j) \in \mathcal{E}, \ (r,s) \in R_i \times R_j\}. \end{aligned} \tag{6.10}$$

Restricting the Lagrange multipliers as above allows dropping the second minimization on the right hand side of (6.8). Therefore, we obtain

$$q(\lambda) = \sum_h \min_{x^h \in \mathcal{F}^h} \{E^h(x^h) + \lambda^h \cdot x^h\}. \tag{6.11}$$

Noting that (6.2) is a linear function, we let c^h be a vector consisting of $E(i_r)$ and $E(i_r j_s)$ for all rotamers and rotamer pairs so that $E^h(x^h) = c^h \cdot x^h$ for all $x^h \in \mathcal{F}^h$.

Then, finally, our dual problem (6.7) is reduced to

$$\begin{array}{ll} \text{maximize} & \sum_h \min_{x^h \in \mathcal{F}^h} (c^h + \lambda^h) \cdot x^h \\ \text{subject to} & \lambda \in D. \end{array} \tag{6.12}$$

6.3 Dual problem and its LP equivalent

In this section, we attempt to justify the use of decomposition by cyclic graphical models instead of tree decomposition. We first review the results on the equivalence between MAP estimation via tree decomposition and the linear program based on the local marginal polytope. Then, we show similar results for cyclic graph decomposition using similar arguments and proofs as used for tree decomposition.

The MAP probability can be upper-bounded by the convex combination of maximum tree probabilities when the convex combination of the tree distributions is equivalent to the original distribution [Wainwright et al., 2005]. It is also known that the best such upper bound found by minimizing over all feasible tree distributions is equal to the maximum value of its dual problem, a linear program over the local marginal polytope. This result can be rephrased in the context of our dual problem as follows:

Theorem 1. *[Wainwright et al., 2005] When $\mathcal{G}^h$ is a tree for all $\mathcal{G}^h \in \{\mathcal{G}^h\}$, The Lagrangian dual to problem (6.12) is given by the following linear program:*

$$\begin{array}{ll} \textit{minimize} & E(x) = \sum_{i \in V} \sum_{r \in R_i} E(i_r) x_{i_r} + \sum_{(i,j) \in E} \sum_{(r,s) \in R_i \times R_j} E(i_r j_s) x_{i_r j_s} \\ \textit{subject to} & x \in LOCAL(\mathcal{G}), \end{array} \tag{6.13}$$

where the local marginal polytope is defined as:

$$\begin{aligned} LOCAL(\mathcal{G}) = \{\mu \in \mathbb{R}^d_+ \mid & \sum_{r \in R_i} \mu_{i_r} = 1 \quad \forall i \in \mathcal{V}, \\ & \sum_{s \in R_j} \mu_{i_r j_s} = \mu_{i_r} \quad \forall (i,j) \in \mathcal{E}\}, \end{aligned} \tag{6.14}$$

By strong duality, the optimal value of (6.12) is equal to the optimal value of (6.13).

Note that LOCAL($\mathcal{G}$) is equivalent to constraints (B.2)–(B.3). In fact, Theorem 1 can be also shown from the integer linear program (B.1)–(B.6), and by using the following known result on the duality of ILP:

Theorem 2. *[Bertsekas, 1999] Suppose an integer linear program*

$$\begin{aligned} &\textit{minimize} && c \cdot x \\ &\textit{subject to} && a_j \cdot x \le b_j, \quad j = 1, \ldots, r, \\ & && x_i \in X_i, \quad i = 1, \ldots, n, \end{aligned} \tag{6.15}$$

where X_i are given finite subsets of the real line. Let q^* denote the optimal dual cost,

$$q^* = \max_{\mu \ge 0} \min_{x_i \in X_i,\ i=1,\ldots,n} f(x) + \sum_{j=1}^{r} \mu_j (a_j \cdot x - b_j). \tag{6.16}$$

Let $\hat{X}_i$ denote the convex hull of the set X_i, and let $\hat{f}$ be the optimal cost of the relaxation of (6.15), where each X_i is replaced by $\hat{X}_i$,

$$\hat{f} = \min_{\substack{a_j \cdot x \le b_j, \quad j=1,\ldots,r, \\ x_i \in \hat{X}_i, \quad i=1,\ldots,n}} f(x). \tag{6.17}$$

Then, $\hat{f} = q^*$.

In the remaining of the section, we show that the dual problem based on cyclic graph decomposition provides at least as tight relaxation of the integer linear program (B.1)–(B.6) as the LP relaxation based on LOCAL($\mathcal{G}$).

Lemma 9. *Suppose $\{T^l\}$ is a tree decomposition of $\mathcal{G}$ (a subgraph of $\mathcal{G}$), and $\mathcal{G}^h$ contains a clique C such that $|\mathcal{V}(C)| \ge 3$ and $i \in \mathcal{V}^h$ for all $i \in C$, and $(i,j) \in \mathcal{E}^h$ for all $(i,j) \in C$. Then, we have the following inclusion property:*

$$\mathcal{M}(\mathcal{G}; \mathcal{G}^h) \subset \bigcap_l LOCAL(\mathcal{G}; T^l) \cap \mathcal{C}(\mathcal{G}^h; C), \tag{6.18}$$

where

$$\mathcal{M}(\mathcal{G};\mathcal{G}^h) = \{\mu \in \mathbb{R}_+^{d(\mathcal{G})} \mid \exists p(\mathbf{x}) \ s.t. \ p(x_i = r) = \mu_{i_r} \ \ \forall i \in \mathcal{V}^h$$
$$p(x_i = r, x_j = s) = \mu_{i_r j_s} \ \ \forall (i,j) \in \mathcal{E}^h\}, \tag{6.19}$$

$$LOCAL(\mathcal{G};T^l) = \{\mu \in \mathbb{R}_+^{d(\mathcal{G})} \mid \sum_{r \in R_i} \mu_{i_r} = 1 \ \ \forall i \in \mathcal{V}(T^l),$$
$$\sum_{s \in R_j} \mu_{i_r j_s} = \mu_{i_r} \ \ \forall (i,j) \in \mathcal{E}(T^l)\}, \tag{6.20}$$

and

$$\mathcal{C}(\mathcal{G};\mathcal{G}^h, C) = \{\mu \in \mathbb{R}^{d(\mathcal{G})}, \pi \in \mathbb{R}^{d(C)} \mid \sum_{\{x'_C \mid (x'_i, x'_j) = (r,s)\}} \pi(x'_C) = \mu_{i_r j_s} \tag{6.21}$$
$$\forall (i,j) \in \mathcal{E}(C) \cap \mathcal{E}(G^h)\}, \tag{6.22}$$

In problem (6.12), if we let

$$\Theta = \{\{\theta^h\} \in \mathbb{R}^{\tilde{d}} \mid \sum_{h: i \in \mathcal{G}^h} \theta^h_{i_r} = c_{i_r} \ \ \forall i \in \mathcal{V}, r \in R_i, \tag{6.23}$$
$$\sum_{h:(i,j) \in \mathcal{E}^h} \theta^h_{i_r j_s} = c_{i_r j_s} \ \ \forall (i,j) \in \mathcal{E}, (r,s) \in R_i \times R_j\}, \tag{6.24}$$

then we can easily verify that there is a one-to-one correspondence between Θ and D because for any $\{\lambda^h\} \in D$ and $\{\theta^h\} = \{c^h + \lambda^h\}$, we have

$$\sum_{h: i \in \mathcal{G}^h} \theta^h_{i_r} = \sum_{h: i \in \mathcal{G}^h} (c^h_{i_r} + \lambda^h_{i_r}) = \sum_{h: i \in \mathcal{V}^h} c^h_{i_r} = c_{i_r}, \tag{6.25}$$

for all $i \in \mathcal{V}$, $r \in R_i$, and

$$\sum_{h:(i,j) \in \mathcal{E}^h} \theta^h_{i_r j_s} = \sum_{h:(i,j) \in \mathcal{E}^h} (c^h_{i_r j_s} + \lambda^h_{i_r j_s}) = \sum_{h:(i,j) \in \mathcal{E}^h} c^h_{i_r j_s} = c_{i_r}, \tag{6.26}$$

for all $(i,j) \in \mathcal{E}$, $(r,s) \in R_i \times R_j$. Therefore, the following problem is equivalent to

(6.12):

$$\begin{array}{ll} \text{maximize} & \sum_h \min_{x^h \in \mathcal{F}^h} \theta^h \cdot x^h \\ \text{subject to} & \{\theta^h\} \in \Theta. \end{array} \tag{6.27}$$

For notational simplicity, we define $\phi(\{\theta\}) \in \mathbb{R}^{d(G)}$ such that

$$\phi(\{\theta^h\})_{i_r} = \sum_{h:i\in\mathcal{G}^h} \theta^h_{i_r}, \tag{6.28}$$

and

$$\phi(\{\theta^h\})_{i_rj_s} = \sum_{h:(i,j)\in\mathcal{E}^h} \theta^h_{i_rj_s}. \tag{6.29}$$

The following lemma is used during the proof of Theorem 3:

Lemma 10.

$$\sup_{\theta^h\in\mathbb{R}^{d(G^h)}} \left\{ \min_{x^h\in\mathcal{F}^h} \theta^h \cdot x^h - \tau \cdot \phi(\{\theta^h\}) \right\} = \begin{cases} 0 & \textit{if } \tau \in \mathcal{M}(\mathcal{G};\mathcal{G}^h), \\ +\infty & \textit{otherwise.} \end{cases} \tag{6.30}$$

Theorem 3. *The Lagrangian dual of problem (6.27) is given by the following linear program:*

$$\begin{array}{ll} \textit{minimize} & E(x) = \sum_{i\in V}\sum_{r\in R_i} E(i_r)x_{i_r} + \sum_{(i,j)\in E}\sum_{(r,s)\in R_i\times R_j} E(i_rj_s)x_{i_rj_s} \\ \textit{subject to} & x \in \bigcap_h \mathcal{M}(\mathcal{G};\mathcal{G}^h). \end{array} \tag{6.31}$$

By strong duality, the optimal value of (6.31) is equal to the optimal value of (6.27).

We obtain the following inequalities regarding the optimal value of problem (C.44):

Lemma 11.

$$\min_{\tau\in LOCAL(\mathcal{G})} \tau \cdot c \leq \min_{\tau\in LOCAL(\mathcal{G})\bigcap_h \mathcal{C}(\mathcal{G}^h;C^h)} \tau \cdot c \leq \min_{\tau\in\bigcap_h \mathcal{M}(\mathcal{G};\mathcal{G}^h)} \tau \cdot c, \tag{6.32}$$

where C^h is some clique contained by $\mathcal{G}^h$ for each h.

The inequality between the first term and the third term of (6.32) suggests that the optimal value of (6.12) is greater than or equal to the optimal value of (6.13). However, this does not tell us how much improvement in the tightness of the lower bound is gained. On the other hand, the results for tree decomposition state that the optimal value of tree-reweighted relaxation does not depend on the set of trees used for decomposition only if the set of trees cover the original graphical model. For decomposition by cyclic graphs, we do not have such a result, but the inequality (6.32) suggests that we can modestly separate the effect of using cliques in the decomposition; addition of cliques will always improve the lower bound $\min_{\tau \in \text{LOCAL}(\mathcal{G}) \bigcap_h \mathcal{C}(\mathcal{G}^h; C^h)} \tau \cdot c$ of the optimal value of problem (6.12) regardless of how the cliques are distributed among the subgraphs.

6.4 Solving the dual problem by the subgradient method

The dual problem (6.12) we obtained above is nondifferentiable with respect to the Lagrange multipliers λ, but can be solved by the standard dual methods such as the subgradient method and the cutting plane method. The subgradient method uses iteration of two stage optimization where the subproblems are solved for given Lagrange multipliers, then the Lagrange multipliers are updated based on the solution of the subproblems. The cutting plane method instead attempts to find the optimal Lagrange multipliers by constructing a series of polyhedral approximations of the dual function. In this method, the polyhedral approximation is repeatedly refined based on the solution of the linear program, that is, the previous polyhedral approximation. We are going to use the subgradient method in this work because it is simple and intuitive, and works well in practice for our problem. Several drawbacks of the cutting plane method are the need of a linear program solver, additional computation for finding the subgradient separately from solving the linear program, and its known difficulty with convergence.

The key idea of the subgradient method is generating a sequence of Lagrange multipliers that eventually converges to the optimal dual solution. A general description of the subgradient method is as follows:

1. Assign initial values to the Lagrange multipliers.
2. Find the subgradient for the Lagrange multipliers by solving the subproblems.
3. Compute a stepsize based on the subgradient and the current objective value.
4. Update the Lagrange multipliers based on the stepsize and the subgradient.
5. If the Lagrange multipliers did not converge, go back to step 2.

In the following, we will elaborate the technical details of steps 2 to 4 of the subgradient method outlined above using our dual problem (6.12).

6.4.1 Finding a subgradient

The subgradient is an analog of the gradient, and is defined for nondifferentiable concave (or convex) functions. In general, $g(\lambda)$ is a subgradient for a concave function $q(\lambda)$ at λ if

$$q(\bar{\lambda}) \leq q(\lambda) + (\bar{\lambda} - \lambda) \cdot g(\lambda), \quad \forall \bar{\lambda} \in \mathbb{R}^d. \tag{6.33}$$

Such a subgradient can be immediately obtained by solving the subproblems. In our dual problem, if we let

$$x^h(\lambda^h) \in \arg\min_{x^h \in F^h} (c^h + \lambda^h) \cdot x^h, \tag{6.34}$$

for each h[1], and

$$\nabla q(\lambda) = \{x^h(\lambda^h)\} \in \mathbb{R}^{\tilde{d}}, \tag{6.35}$$

[1]In case there are many values of $x^h \in F^h$ that attain the minimum of $(c^h + \lambda^h) \cdot x^h$, one of them is arbitrarily selected as $x^h(\lambda^h)$.

then we have

$$\begin{aligned} q(\bar{\lambda}) &= \sum_h \min_{\{x^h \in F^h\}} (c^h + \bar{\lambda}^h) \cdot x^h \\ &\leq \sum_h (c^h + \bar{\lambda}^h) \cdot x^h(\lambda^h) \\ &= \sum_h (c^h + \lambda^h) \cdot x^h(\lambda^h) + \sum_h (\bar{\lambda}^h - \lambda^h) \cdot x^h(\lambda^h) \\ &= q(\lambda) + (\bar{\lambda} - \lambda) \cdot \nabla q(\lambda). \end{aligned} \tag{6.36}$$

Therefore, $\nabla q(\lambda)$ is a subgradient. We will discuss the solution method of subproblems using tree decomposition and dynamic programming in Section 6.5.

Although the conventional subgradient method requires exact solution of all subproblems to find a subgradient, there exist variants of the subgradient method such as surrogate gradient algorithms [Zaho et al., 1999], conditional ϵ-subgradient methods [Larsson et al., 2003], and incremental subgradient methods [Nedic and Bertsekas, 2001] that use approximate solution of a subset of subproblems and still guarantee convergence under certain conditions. In this work, we consistently use the conventional subgradient method in solving the subproblems, and leave application of these variants to our problem as a future work. Instead, we focus on finding a tight Lagrangian dual relaxation through addition of triplets or cuts.

6.4.2 Updating the Lagrange multipliers

The rule for generating the $(k+1)^{th}$ Lagrange multipliers from the k^{th} ones is

$$\lambda^{(k+1)} = \left[\lambda^{(k)} + s^{(k)} \nabla q^{(k)}(\lambda)\right]^+, \tag{6.37}$$

where $s^{(k)} \in \mathbb{R}_+$ is a stepsize, and $[\cdot]^+$ denotes projection onto D, the domain of λ. Considering the form of D (6.10) and $\nabla q(\lambda)$ (6.34)–(6.35), the projection can be implemented as follows [Komodakis et al., 2007]:

$$\lambda_{i_r}^{h,(k+1)} = \lambda_{i_r}^{h,(k)} + s^{(k)} \left\{ x^{h,(k)}(\lambda^{(k)})_{i_r} - \frac{\sum_{h:i\in\mathcal{V}^h} x^{h,(k)}(\lambda^{(k)})_{i_r}}{|\{h \mid i \in \mathcal{V}^h\}|} \right\} \tag{6.38}$$

$$\lambda_{i_r j_s}^{h,(k+1)} = \lambda_{i_r j_s}^{h,(k)} + s^{(k)} \left\{ x^{h,(k)}(\lambda^{(k)})_{i_r j_s} - \frac{\sum_{h:(i,j)\in\mathcal{E}^h} x^{h,(k)}(\lambda^{(k)})_{i_r j_s}}{|\{h \mid (i,j) \in \mathcal{E}^h\}|} \right\}. \tag{6.39}$$

It is straightforward to see that $\lambda^{(k+1)}$ is in D.

It should be noted that the update rule of the subgradient method above does not guarantee improvement of the dual function value at each iteration. Instead, with the use of a proper stepsize rule, the distance from the current Lagrange multipliers to the optimal dual solution

$$\lambda^* = \arg\max_{\lambda\in D} q(\lambda) \tag{6.40}$$

decreases every iteration, that is,

$$\|\lambda^{(k+1)} - \lambda^*\| < \|\lambda^{(k)} - \lambda^*\|. \tag{6.41}$$

6.4.3 Determining the stepsize

The distance to the optimal dual solution is reduced as described by (6.41) if stepsize $s^{(k)}$ satisfies [Bertsekas, 1999]

$$0 < s^{(k)} < \frac{2(q(\lambda^*) - q(\lambda^{(k)}))}{\|\nabla q(\lambda^{(k)})\|^2}. \tag{6.42}$$

Since we do not know $q(\lambda^*)$, a more practical rule takes the form of

$$s^{(k)} = \frac{\alpha^{(k)}(q^{(k)} - q(\lambda^{(k)}))}{\|\nabla q(\lambda^{(k)})\|^2}, \tag{6.43}$$

where $\alpha^{(k)}$ and $q^{(k)}$ can be defined in a number of different ways to guarantee convergence of the subgradient method. For example, $q^{(k)}$ can be the best known upper bound to $q(\lambda^*)$ at the k^{th} iteration, and $\alpha^{(k)} = \frac{1+m}{1+k}$ for some fixed positive integer m. Despite the theoretical convergence guarantee, these types of stepsize rules are very slow to converge in practice. As a result, there have been many theoretical or empirical works that attempt to improve the convergence rate of the subgradient method in different contexts [Bertsekas, 1999, Fumero, 2001].

According to our experiments, although not comprehensive, all the stepsize rules from the literature applied to our dual problem lead to very slow convergence. Instead, the best working stepsize rule found so far on our problem is a simple geometric reduction rule. That is,

$$s^{(k+1)} = \begin{cases} s^{(k)} & \text{if } q(\lambda^{(k)}) < q(\lambda^{(k-1)}), \\ \beta s^{(k)} & \text{otherwise,} \end{cases} \tag{6.44}$$

for some $0 < \beta < 1$. For $\beta \approx 1$, e.g. 0.995, the subgradient method often ended up with fairly accurate estimation of the dual optimal value, while smaller β such as 0.95 may lead to fast but premature convergence. Still, $\beta = 0.95$ found better dual values that all other stepsize rules we tried.

The term "convergence" used in most subgradient literature refers to the convergence to the dual optimal solution. However, as discussed above, it is rare that such convergence is attained by applying the subgradient method with known stepsize rules to large-scale dual problems. When the subgradient method is used with a geometric stepsize rule like (6.44), possible stopping criteria are

$$||\lambda^{(k+1)} - \lambda^{(k)}|| < \epsilon, \tag{6.45}$$

for some very small $\epsilon > 0$, or

$$|E(\tilde{x}) - q(\lambda^{(k)})| < \epsilon, \tag{6.46}$$

where $\tilde{x}$ is a feasible primal solution, and $E(\tilde{x})$ is, therefore, an upper bound to the dual optimal value. In our dual problem, we can construct $\tilde{x}$ by using a solution to one of the subproblems or by combining solutions for multiple subproblems as described in Section 2.4.1.

6.5 Solving the subproblems

In Section 6.3, it was shown that we can benefit from using decomposition with cyclic graphical models instead of using trees when stronger lower bounds are required.

A problem with using cyclic graph decomposition is that it may involve solution of non-trivial subproblems. It was shown that such a subproblem can be exactly solved by finding a tree decomposition [Robertson and Seymour, 1986] (also known as junction tree [Cowell et al., 1999], join tree, or clique tree) of the graphical model into a junction tree, and then applying dynamic programming to the tree decomposition [Dawid, 1992]. The running time of such an algorithm is exponential to the size of the largest clique in the junction tree, which is called treewidth. Although the exponential running time is unacceptable in general, by constraining the treewidth of subgraphs used in decomposition, we may obtain reasonable solution time in practice. Therefore, the choice of decomposition by cyclic subgraphs should be made to balance the running time determined by treewidth, and the strength of lower bounds obtained by solving the dual problem as discussed in Section 6.3.

The remainder of this section briefly reviews the notion of tree decomposition and treewidth, and the dynamic programming method applied to tree decomposition.

6.5.1 Tree decomposition

The notion of tree decompositions and treewidth was introduced by Robertson and Seymour [1986]. A tree decomposition is a mapping of a graph into a tree that can be used to solve certain combinatorial problems defined over the original graph.

Definition 1. *A tree decomposition of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a pair (T, X), where $T = (I, F)$ is a tree with a node set I and an edge set F, and $X = \{X_i \mid i \in I,\ X_i \subset \mathcal{V}\}$, satisfying the following properties:*

1. *each vertex $v \in \mathcal{V}$ belongs to at least one X_i (a bag of i), $i \in I$;*

2. *for each edge $(i, j) \in \mathcal{E}$, there is $i \in I$ such that $i \in X_i$ and $j \in X_i$;*

3. *for all $v \in \mathcal{V}$, the set of nodes $\{i \in I \mid v \in X_i\}$ induces a subtree of T.*

The width of a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is defined as $\max_{i \in I} |X_i| - 1$. The treewidth of a graph $\mathcal{G}$ is the minimum width over all tree decompositions of $\mathcal{G}$.

By the definitions above, every tree has treewidth one. Any graph containing a cycle has tree width at least two because condition 3 of Definition 1 forces at least three vertices to belong to some X_i, $i \in I$.

In general, it is NP-complete to decide whether the treewidth of a given graph is at most a given variable k [Arnborg et al., 1987]. However, there exists a linear time algorithm to determine whether a given graph has treewidth at most a given constant k and to find such a tree decomposition [Bodlaender, 1996]. However, the constant factor in the running time of the algorithm is known to be too large for the algorithm to be practical. There are also other exact algorithms [Brodlaender et al., 2006, Fomin et al., 2004], and approximation algorithms [Brodlaender et al., 1995, Feige et al., 2005], and heuristic algorithms [Tarjan and Yannakakis, 1984, Rose, 1972, Kjaerulff, 1990]. In solving the dual problem, we need to perform tree decomposition for each subproblem only once at the start. Therefore, the running time of a tree decomposition algorithm may not be the highest concern in solving the dual problem. On the other hand, because the number of rotamers for each variable varies, treewidth may not exactly represent the complexity. Instead, tree decomposition that attains as small $\max_{i \in I} \sum_{v \in X_i} |R_v|$ as possible can be more desirable.

6.5.2 Dynamic programming on tree decompositions

Our subproblem in (6.11) has the form of

$$\begin{aligned} \text{minimize} \quad & E^h(x^h) + \lambda^h \cdot x^h \\ \text{subject to} \quad & x^h \in \mathcal{F}^h. \end{aligned} \tag{6.47}$$

For notational purpose in this section, this will be rewritten in the form of our original GMEC problem as follows:

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) = \sum_{i\in\mathcal{V}} f_i(x_i) + \sum_{(i,j)\in\mathcal{E}} f_{ij}(x_i, x_j) \\ \text{subject to} & \qquad \mathbf{x} = (x_1, \ldots, x_n),\ x_i \in R_i, \end{array} \tag{6.48}$$

where the discrete cost function is defined over a graphical model. Suppose we obtain a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ for the graphical model $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of (6.48). Without loss of generality, we can assume there exists a root node $\Upsilon \in I$ of the tree decomposition. Then, we can define the following notations for $i \in I$:

- $pa(i)$: i's parent for $i \neq \Upsilon$.
- $D(i)$: the set of descendants of i.
- $depth(i)$: depth of i, computed as 1 if $i = \Upsilon$, and $depth(pa(i)) + 1$ otherwise.
- $\Gamma(i)$: the set of neighbors of i in T, that is $pa(i) \cup D(i)$.
- $S(i)$: the set of $v \in \mathcal{V}$ such that the depth of $i \in T$ is the smallest among all $j \in I$ such that $v \in X_j$.
- $P(i)$: the set of $(v, w) \in \mathcal{E}$ such that the depth of i in T is the smallest among all $j \in I$ such that $(v, w) \in X_j$.
- $T(i)$: subtree of T consisting of i and its descendants.

Suppose a separating vertex set $S \subset \mathcal{V}$ separates $\mathcal{V}$ into two disjoint sets $\mathcal{V}_1$ and $\mathcal{V}_2$ so that any path from $v_1 \in \mathcal{V}_1$ to $v_2 \in \mathcal{V}_2$ passes through a vertex in S. If we use the notation $\mathcal{G}[W] = (W, \{(v, w) \mid (v, w) \in \mathcal{E},\ v, w \in W\})$ for $W \subset \mathcal{V}$, $\mathcal{V}_1$ and $\mathcal{V}_2$ satisfy $\mathcal{G}[\mathcal{V}\backslash S] = \mathcal{G}[\mathcal{V}_1] \cup \mathcal{G}[\mathcal{V}_2]$. Then, the key idea of dynamic programming is that we can reduce the subproblem (6.48) to two smaller problems over $\mathcal{G}[\mathcal{V}_1]$ and $\mathcal{G}[\mathcal{V}_2]$ once the optimal assignment for $v \in S$ is fixed. Lemma 12 shows that tree decomposition provides such a separating vertex set.

Lemma 12. *Given a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ for $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Then, removing X_Υ disconnects $\mathcal{G}$ so that*

1. *any $v \in V \backslash X_\Upsilon$ can appear in at most one subtree of $T \backslash s$;*
2. *each subtree of T defines at least one component.*

Since the selection of Υ in Lemma 12 is arbitrary, any X_i, $i \in I$ is a separating vertex set of $\mathcal{G}$.

Lemma 13 is useful for the proof of Lemma 14.

Lemma 13. *The following properties are satisfied for a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$:*

1. *$\bigcup_{i \in I} S(i) = \mathcal{V}$ and $S(i) \cap S(j) = \phi$ for all $i, j \in I, i \neq j$.*
2. *$\bigcup_{i \in I} P(i) = \mathcal{E}$ and $P(i) \cap P(j) = \phi$ for all $i, j \in I, i \neq j$.*

By Lemma 13, $f(\mathbf{x})$ can be decomposed as follows:

$$f(\mathbf{x}) = \sum_{u \in I} f^u(\mathbf{x}_u), \tag{6.49}$$

where $\mathbf{x}^u$ is the set of variables in X_u, and

$$f^u(\mathbf{x}^u) = \sum_{i \in S(u)} f_i(x_i) + \sum_{(i,j) \in P(u)} f_{ij}(x_i, x_j). \tag{6.50}$$

We also use the notation indexed by a set, such as, for $A \in I$,

$$f^A(\mathbf{x}^A) = \sum_{u \in A} f^u(\mathbf{x}^u). \tag{6.51}$$

The following lemma suggests that once the assignment for node Υ is fixed, the minimization problem can be solved separately on each subtree.

Lemma 14.

$$\min_{\{\tilde{\mathbf{x}} | \tilde{\mathbf{x}}^\Upsilon = \mathbf{x}^\Upsilon\}} f(\mathbf{x}) = f^\Upsilon(\mathbf{x}^\Upsilon) + \sum_{i \in D(\Upsilon)} \omega^i(\mathbf{x}^{\Upsilon \cap i}), \tag{6.52}$$

where

$$\omega^i(\mathbf{x}^{\Upsilon\cap i}) = \min_{\{\tilde{\mathbf{x}}^{T(i)} | \tilde{\mathbf{x}}^{\Upsilon\cap i} = \mathbf{x}^{\Upsilon\cap i}\}} f^{T(i)}(\tilde{\mathbf{x}}^{T(i)}). \tag{6.53}$$

Based on Lemma 14, the dynamic programming algorithm for our subproblem can be described as in Algorithm 4. The dynamic programming is a two-step procedure, that is, `Bottom-to-Top` and `Top-to-Bottom`, where each procedure is described recursively in Algorithm 5 and 6. In the `Bottom-to-Top` procedure, given the cost of child nodes, the best cost for each assignment of variables in the intersection of the current node u and its parent $pa(u)$ are computed, that is, $\omega^u(\mathbf{x}^{pa(u)\cap u})$. Then, in `Top-to-Bottom` procedure, given the optimal assignment for the intersection of the current node u and its parent $pa(u)$, the optimal assignment for variables in $u \backslash pa(u)$ is searched using the pre-computed optimal cost $\omega^i(\mathbf{x}^{u\cap i})$ for all $i \in I$, $|D(i)| > 0$.

Algorithm 4: Dynamic programming algorithm for the subproblem based on tree decomposition. The algorithm is a two-step procedure: `Bottom-to-Top` and `Top-to-Bottom`, where each procedure is described recursively in Algorithm 5 and 6.

```
  Data: $f(\mathbf{x})$, $(\{X_i \mid i \in I\}, T = (I, F))$
  Result: $\arg\max_{\mathbf{x}} f(\mathbf{x})$
1 begin
2     Bottom-to-Top($\Upsilon$)
3     Top-to-Bottom($\Upsilon$)
4 end
```

Algorithm 5: `Bottom-to-Top` procedure. Given the cost of child nodes, the best cost for each assignment of variables in the intersection of the current node u and its parent $pa(u)$ are computed, that is, $\omega^u(\mathbf{x}^{pa(u)\cap u})$.

```
  Data: $u \in I$
  Result: $\omega^u(\mathbf{x}^{pa(u)\cap u})$
1 begin
2     foreach $i \in D(u)$ do
3         Bottom-to-Top($i$)
4     if $u \neq \Upsilon$ then
5         Compute $\omega^u(\mathbf{x}^{pa(u)\cap u})$ for all values of $\mathbf{x}^{pa(u)\cap u}$
6 end
```

Algorithm 6: `Top-to-Bottom` procedure. Given the optimal assignment for the intersection of the current node u and its parent $pa(u)$, the optimal assignment for variables in $u\ pa(u)$ is searched using the pre-computed optimal cost $\omega^i(\mathbf{x}^{u \cap i})$.

```
  Data: $u$, $\mathbf{x}^{u \cap pa(u)}$
  Result: $\mathbf{x}^{u \setminus pa(u)}$
1 begin
2     Compute $\arg\min_{\mathbf{x}^{u \setminus pa(u)}} f^u(\mathbf{x}^u) + \sum_{i \in D(u)} \omega^i(\mathbf{x}^{u \cap i})$
3     foreach $i \in D(u)$ do
4         Top-to-Bottom($i$, $\mathbf{x}^{i \cap u}$)
5 end
```

6.6 Computational results

6.6.1 Implementation

We implemented the subgradient method in C++ to evaluate its speed and bounding strength on protein design cases. Some implementation details are discussed below.

Subproblem solver

Subgradient method for MAP estimation was implemented using dynamic programming for the solution of subproblems. Each subproblem is a MAP estimation problem over a graphical model with small treewidth. The dynamic programming uses library `TreeDecomposer` [Subbarayan and Andersen] for tree decomposition, then runs `Bottom-to-Top` and `Top-to-Bottom` procedures. To save memory, the optimal assignment of each non-intersecting region was not saved during `Bottom-to-Top`. Instead, this is found during `Top-to-Bottom` using the optimal scores from `Bottom-to-Top`.

Dual parameters update

Each iteration of the subgradient method requires update of only a small set of singleton and pairwise parameters of each subproblem. The updated parameters correspond to the rotamers and rotamer pairs that were selected as a solution of some subproblem. Therefore, the maximum number of updated singleton parameters at each iteration is

$$\sum_{v \in \mathcal{V}} |\{h \mid v \in \mathcal{V}^h\}|. \tag{6.54}$$

Similarly, the maximum number of updated pairwise parameters is

$$\sum_{e \in \mathcal{E}} |\{h \mid e \in \mathcal{E}^h\}|. \tag{6.55}$$

The actual number of updated parameters can be smaller because subproblems may share solution for vertices or edges. For example, when $(i, j) \in \mathcal{E}^{h_1}$ and $(i, j) \in \mathcal{E}^{h_2}$, the rotamer pair (i_r, j_s) selected as the solution of subproblem h_1 can be also the solution for the same edge in h_2. Exploiting this fact in the subgradient method dramatically reduces the running time.

Stepsize rules

We experimented with different stepsize rules. Stepsize rules tested are

1. pure subgradient $g^{(k)}$ with the geometric stepsize rule of (6.44),

2. Fumero's rule [Fumero, 2001],

3. modified subgradient $d^{(k)} = g^{(k)} + \gamma d^{(k-1)}$ with stepsize $(ub_{best} - lb_{curr})/\|d^{(k)}\|^2$ (ub_{best} is the best known upper bound, and lb_{curr} is the lower bound from the current iteration),

4. modified subgradient $d^{(k)} = g^{(k)} + \gamma d^{(k-1)}$ with stepsize $(ub_{best} - lb_{curr})/\|g^{(k)}\|^2$,

with variations of the following options:

1. different values for ν_1, ν_2, ϵ_0, r_1, and γ [Fumero, 2001].

2. different increment/decrement of β [Fumero, 2001].

3. modifying the subgradient direction so that an acute angle is maintained between the current and the previous subgradients.

4. backing up the best Lagrange multipliers $\lambda^{(k)}$ and restoring it after a number of unimproving updates.

Among different rules and variations we tried to improve the strength of bounds and running times based on previous theoretical and empirical work, most stepsize rules performed worse than the geometric stepsize rule. When considering only the strength of lower bounds, the best performing step size rule was the geometric stepsize rule, combined with $d^{(k)} = g^{(k)} + 0.5d^{(k-1)}$, and parameters back-up/restoration. However, using such a complicated updating rule is not worth the increase of running time compared to the gain in the tightness of lower bounds. Note that the step size rule $\beta^k(ub_{best} - lb_{curr})/\|d^{(k)}\|^2$ was not tried in our experiment, which later was found commonly adopted by much previous work.

Parameters and Environment

In the following experiments, we consistently used the stepsize rule of (6.44) with $\beta = 0.995$. The stopping condition used is $\|\Delta\lambda\|_\infty < 10^{-7}$. Tests were run on a Linux workstation with a Dual-Core AMD Opteron Processor 2214, and 2 GBytes of memory. Only a single core was used in each experiment. The C++ code was compiled with GNU compiler version 3.3.5.

6.6.2 Triplet selection by the strength of interactions

We used triplets as the basic components of subgraphs. That is, the maximum treewidth of a subgraph is limited to two. The heuristic selection of triplets were done as follows:

1. compute a measure of interactions for each $e \in \mathcal{E}$ (denoted as ϖ_e).

2. collect all triplets such that ϖ_e of all e in the triplet is greater than a cutoff value, C_{IQR}.

3. compute scores of each triplet from above as the sum of ϖ_e of all e in the triplet.

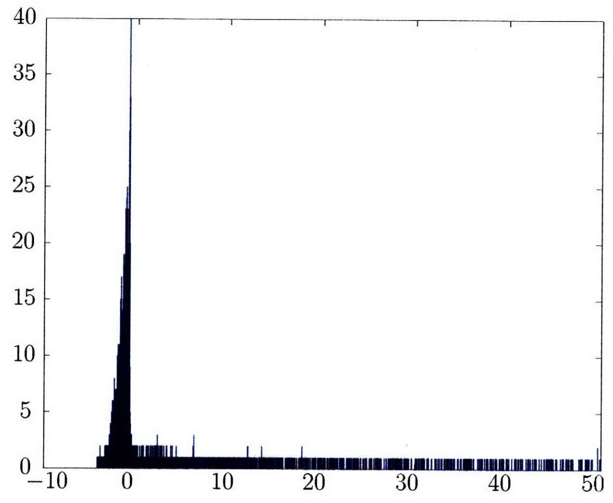


Figure 6-1: Histogram of pairwise energies between position 3 and 4 of a subproblem of SWEET7-NOW. The standard deviation is not an adequate measure of the dispersion due to the high peak close to 0 and the wide range of the distribution.

4. Sort the triplets by the score.

5. Pick a specified number of triplets with largest scores.

The strength measure ϖ used here is dispersion of pairwise energies, called the interquartile range (IQR). IQR is the difference between the third and first quartiles. IQR is a robust alternative to the standard deviation, and is less affected by extremes. Figure 6-1 shows a typical distribution of pairwise energies between a specific pair of positions. It shows that a dominant number of pairwise energies are close to 0 whereas the total range (the difference between the minimum and the maximum values) of pairwise energies is fairly wide. This suggests that the standard deviation will be very small compared to the range, and fail to convey the actual dispersion of the distribution. On the other hand, the total range is also not an adequate measure because it gives equal emphasis to every value in the range including extremely large values, which are not likely to be included in the solution of the optimization problem and therefore do not affect the solution procedure.

The triplets were used to construct subgraphs as follows:

1. Determine N_{tri} triplets to add.
2. Set the maximum treewidth tw_{max} of subgraphs to be constructed.
3. Construct a set of spanning trees that cover the original graph.
4. For each tree from above, add triplets while the total number of triplets is less than N_{tri} and the resulting treewidth is less than equal to tw_{max}.
5. Repeat step 4 until the total number of added triplets is less than N_{tri}.
6. If the total number of added triplets is less than N_{tri} and no more triplets can be added to the spanning trees because of the treewidth restriction, construct a new subgraph consisting of only of a single triplet.
7. Add more triplets to the new subgraph from step 6 while the total number of triplets is less than N_{tri} and the resulting treewidth is less than equal to tw_{max}. Only add triplets that will constitute a connected subgraph by adding necessary edges from the original graph.
8. Go back to 6 if the total number of added triplets is less than N_{tri}.

In the following, we perform a computational experiment with the subgradient method to observe the change of bounds and the bounding times, that is, the running time of the subgradient method when different numbers of triplets are added.

We applied our implementation of the subgradient method to a small subproblem SWEET7-NOW-51-HIGH of test case SWEET7-NOW. The test case SWEET7-NOW-51-HIGH contains 21 residue positions, 390 rotamers, and 71,385 rotamer pairs. The IQR's of pairwise energies for SWEET7-NOW-51-HIGH are shown in the histogram of Figure 6-2. The median IQR is 0.12, and four position pairs had zero IQR out of total 210 position pairs in the test case. As shown in the histogram, five position pairs have huge IQR values compared to others. Since the irrelevantly high pairwise energies of these position pairs can distort the selection of triplets based on IQR values, we capped IQR's to 100 when computing scores for triplets. The optimal value of the

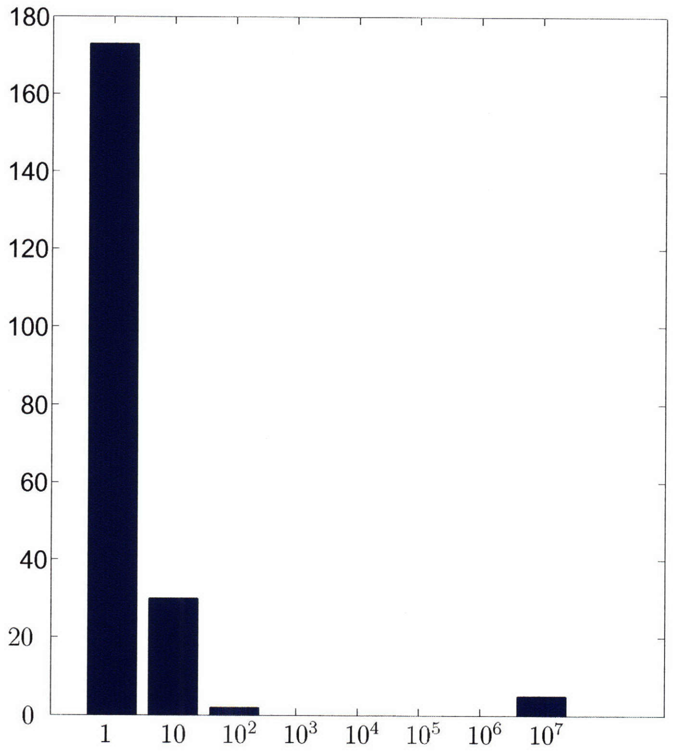


Figure 6-2: Histogram for pairwise energies IQR's of SWEET7-NOW-51-HIGH. The tick values on the x-axis represent the maximum value in the corresponding bin. For example, bin 10^2 contains IQR between 10 and 10^2.

test case is -324.429. TRMP took 5 seconds to compute a lower bound -338.324, and an upper bound -287.585 for this test case.

Figure 6-3 shows the results of computing upper and lower bounds using the subgradient method and the triplets selected by IQR scores. Different curves are derived from different cutoff values for IQR, i.e. C_{IQR} used to select triplets. The plot shows the bounds for at most 50 triplets found for each C_{IQR} value. It is noteworthy that the bounds from the subgradient method are much stronger than that from TRMP. For example, the lower bound from the dual problem derived only from trees and no triplets is -333.812 compared to the TRMP bound of -338.324 although both are supposed to have the same optimal value corresponding to the optimal value of the LP over the local polytope. Such difference is considered due to different algorithmic behaviors, and endows the subgradient method an advantage over TRMP or other similar message-passing algorithms when the quality of bounds matters more than the running time. This is because many diverse numerical strategies can be applied under the framework of the subgradient method in comparison to the fixed formats of message-passing algorithms.

The results shown in Figure 6-3 suggest that using larger C_{IQR} values tends to identify triplets relevant to improving bounds more quickly. However, larger C_{IQR} values lead to fewer eligible position pairs, and therefore fewer triplets. As a result, the curves for larger C_{IQR} end up with smaller total number of triplets and less tight bounds than those for smaller C_{IQR}. Meanwhile, bounds from $C_{IQR} = 0.5$ improve most slowly, but in the end both upper and lower bounds converge to the optimal value by considering triplets with weaker position pairs.

Since IQR can be seen as truncating a large portion of the distribution, we also attempted using a different measure of strength. Figure 6-4 shows the histogram of interoctile ranges (IOR's) of pairwise energies in the same test case. Interoctile range is the difference between 87.5% and 12.5% percentiles. As IOR accounts for wider variations in the data, the histogram of Figure 6-4 slightly spread rightward compared to that of Figure 6-2.

Figure 6-5 shows bounds from the subgradient method as previously done, but triplets were selected by IOR scores instead of IQR scores. As using IOR measure implies more position pairs with extremely large scores, and also more weak edges whose IOR exceeds the cutoff value C_{IOR}, the strong edges tend to be overemphasized by being included in the selected triplets more frequently. As a result, the added triplets have weaker effects on the bounds than the ones selected by IQR scores. The optimal value was not attained by any lower bounds, but tight upper bounds were obtained for all C_{IOR} values.

Figure 6-6 shows the linear relation between the running time of the subgradient method and the number of triplets included in the dual problem. Each data point was computed as an average running time from the experiments described above. It is clear that the stronger bounds from the subgradient method does not come free

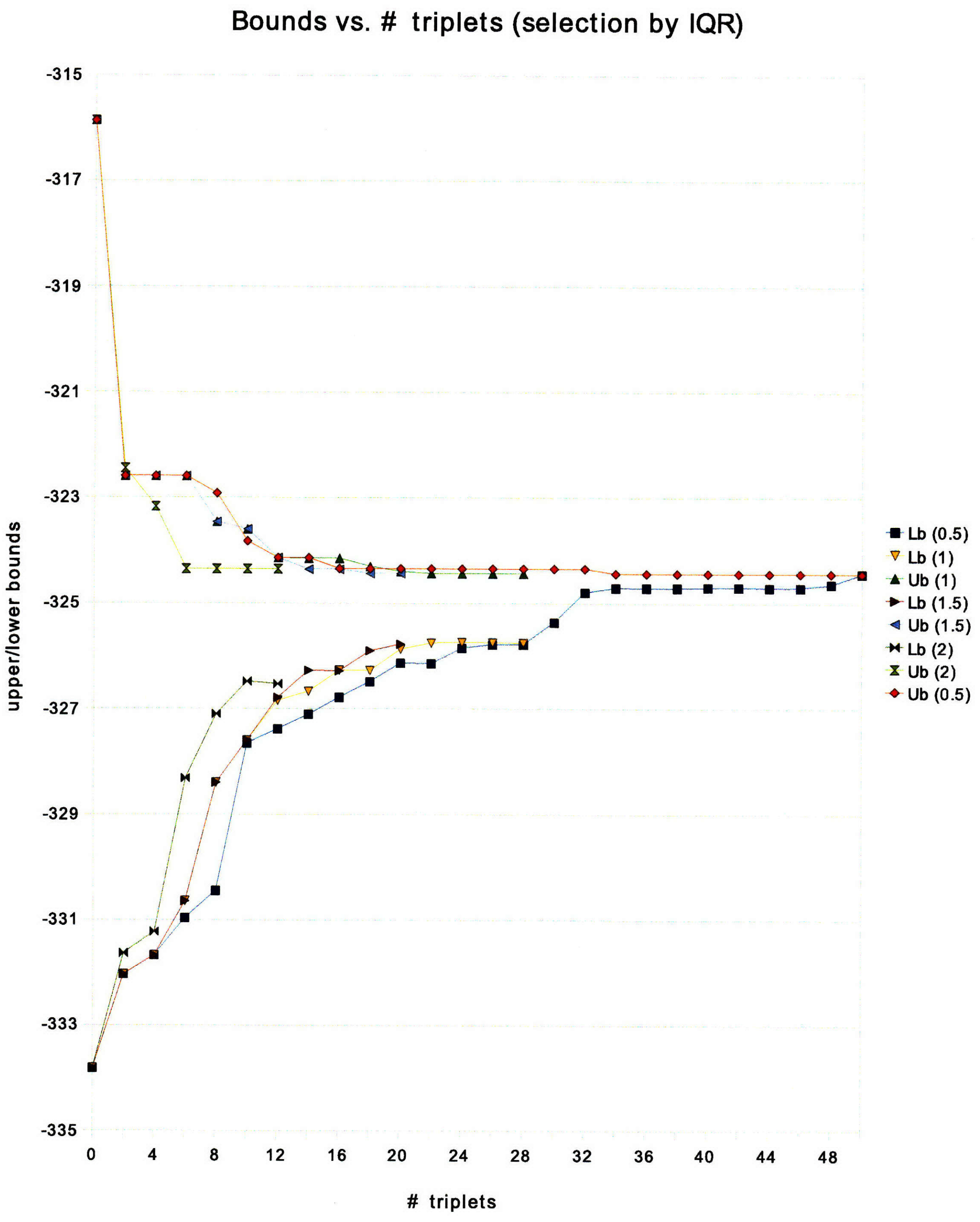


Figure 6-3: Upper and lower bounds from the subgradient method for different numbers of triplets selected by IQR scores for a subproblem of SWEET7-NOW test case. The IQR score was computed as the sum of IQR's of pairwise energies in each triangle. However, all IQR's over 100 was capped at 100. Only those edges whose IQR is over some cutoff value was considered. The cutoff values used are 0.5, 1, 1.5, and 2. The legend show the match between cutoff values and bounds.

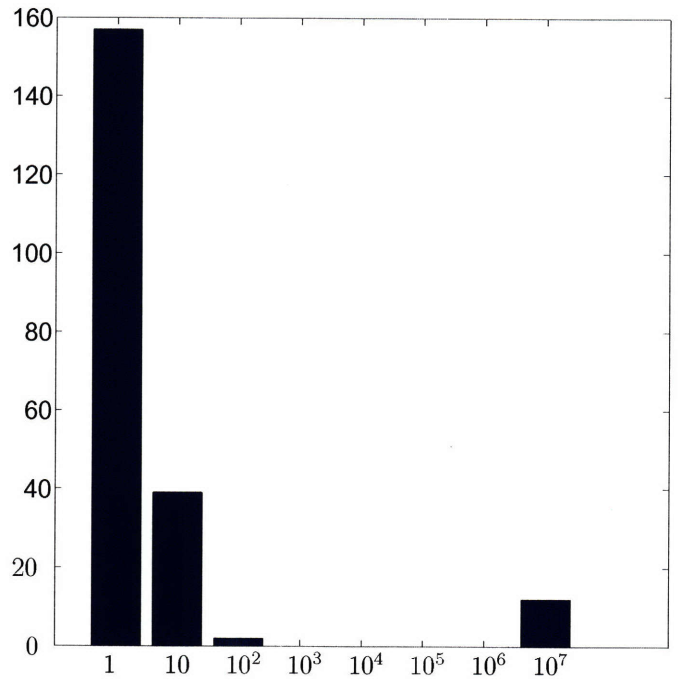


Figure 6-4: Histogram for pairwise energies IOR's of SWEET7-NOW-51-HIGH. The tick values on the x-axis represent the maximum value in the corresponding bin. For example, bin 10^2 contains IQR between 10 and 10^2.

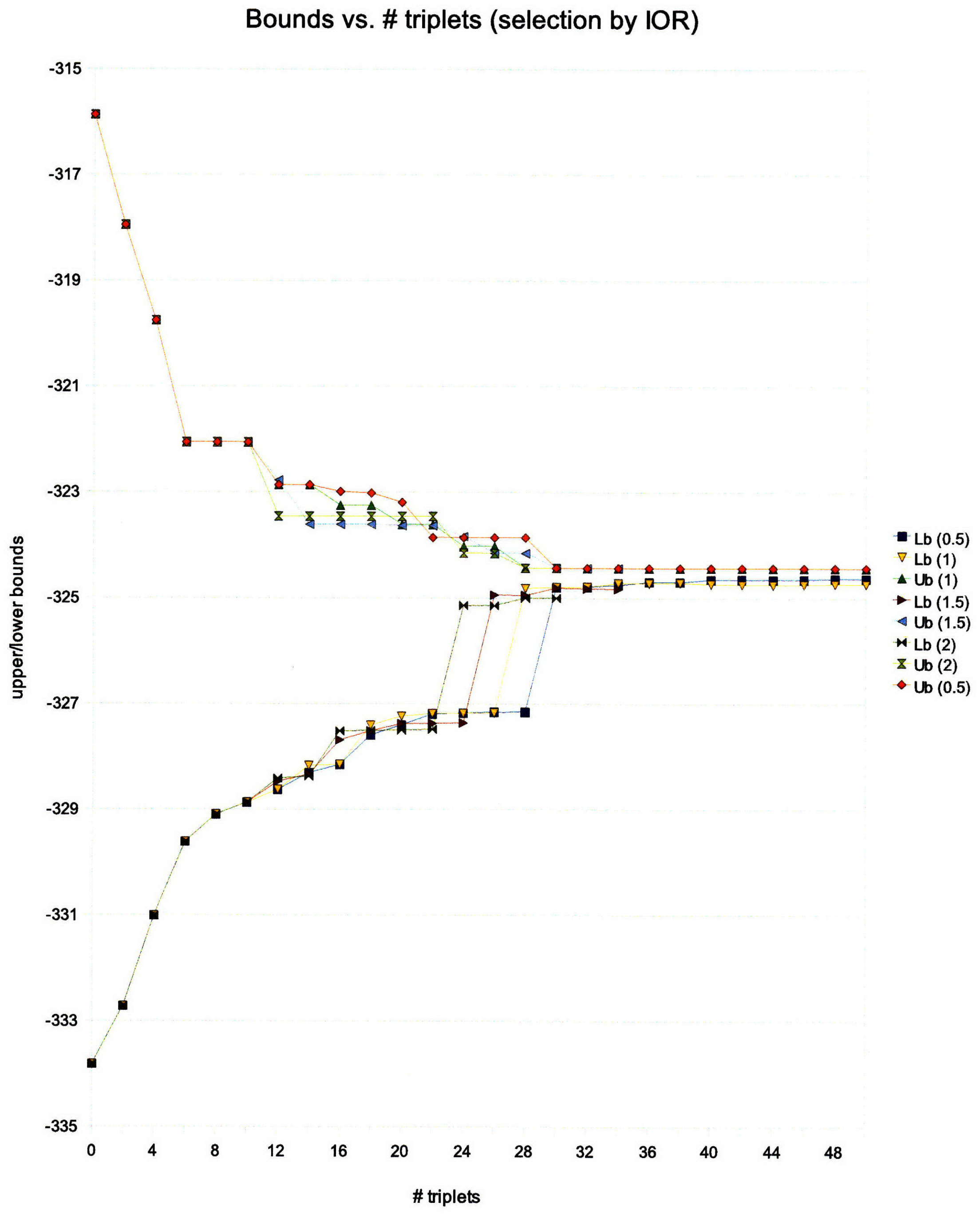


Figure 6-5: Upper and lower bounds from the subgradient method for different numbers of triplets selected by IOR scores for a subproblem of SWEET7-NOW test case. The IOR score was computed as the sum of IOR's of pairwise energies in each triangle. However, all IOR's over 100 was capped at 100. Only those edges whose IOR is over some cutoff value was considered. The cutoff values used are 0.5, 1, 1.5, and 2. The legend show the match between cutoff values and bounds.

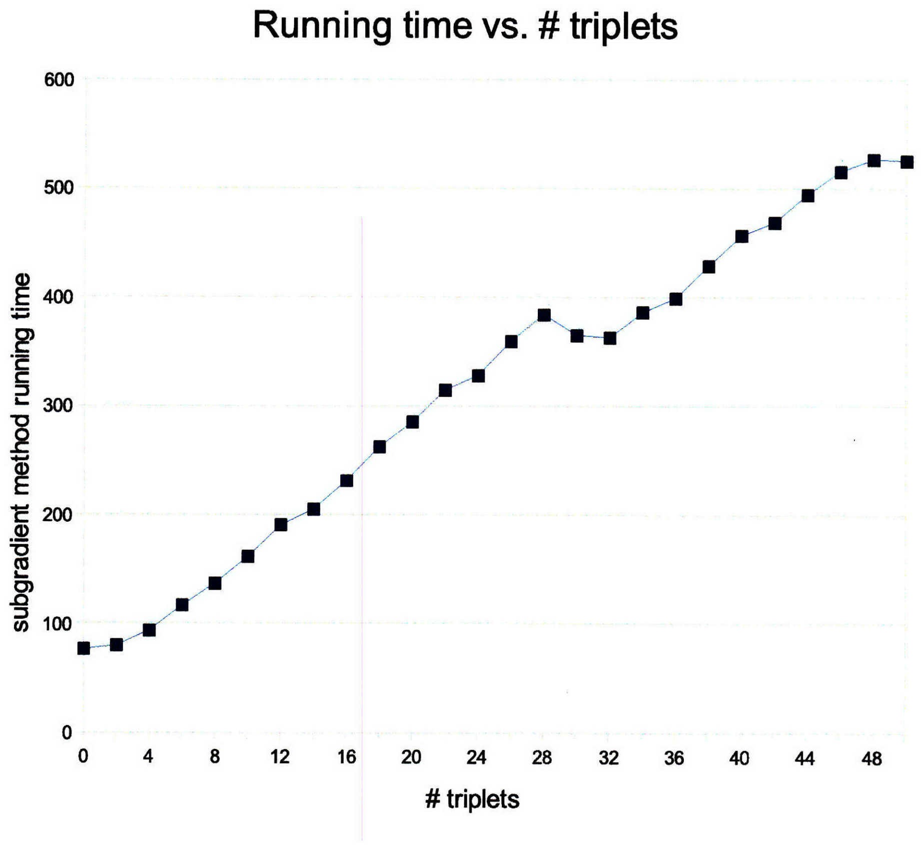


Figure 6-6: Running time of the subgradient method vs. number of triplets included in the dual problem.

considering the running time of TRMP on the same test case is only 5 seconds whereas the subgradient method takes 64 seconds when the dual problem is derived only from trees. However, it should be also noted that the code for the the subgradient method still has room for improvement compared to the highly optimized TRMP code. In addition, adopting more advanced variants of the subgradient method such as the incremental subgradient method [Bertsekas, 1999] is expected to bring further speed-up.

Based on the observations made above, we modify the selection of triplets as follows:

1. set a reasonably large initial value for C_{IQR}.

2. select at most N_{tri} triplets with high IQR scores that satisfy the C_{IQR} cutoff

value, following the procedure suggested previously.

3. done if the number of selected triplets is equal to N_{tri}.

4. otherwise, lower C_{IQR} to a smaller non-negative number, and go back to step 2.

We applied the modified triplets selection scheme and the subgradient method to Case 44 of Table 5.1 preprocessed by DEE. Figure 6-7 shows the resulting upper and lower bounds. The plot also shows bounds from triplets selected by fixed C_{IQR}. Since many position pairs in the test case have IQR's close to 0, we had to set $C_{IQR} = 0$ to collect up to 50 triplets. The bounds from the modified scheme evidently outperform the bounds from the fixed C_{IQR} value, particularly for small numbers of triplets, and also continue to improve as more triplets are added by lowering C_{IQR} values. Note that there are total 55 candidate triplets in the test case, but the gap between the upper and lower bounds are not closed even when 50 triplets are included.

Figure 6-8 shows the selection of triplets from the adaptive lowering of C_{IQR} for Case 44, corresponding to the bounds shown in Figure 6-7. As intended, triplets selected early consist of all strong edges, whereas those selected later also include weak edges.

6.6.3 Triplet selection by disagreement

We tried a different heuristic selection of triplets, which is based on disagreement between subproblem solutions. That is, scores of a triplet was computed as the sum of number of different pairwise solutions of each edge in the triplet. For example, an edge is assigned score 3 if there are total three different choices of rotamer pairs for the edge in the solutions found by solving all subproblems. As a result, the triplet with a high score corresponds to the set of edges with high disagreement between subproblems.

We used the following procedure in our experiment:

1. start with subgraphs with no triplets.

2. solve the dual problem using the subgradient method.

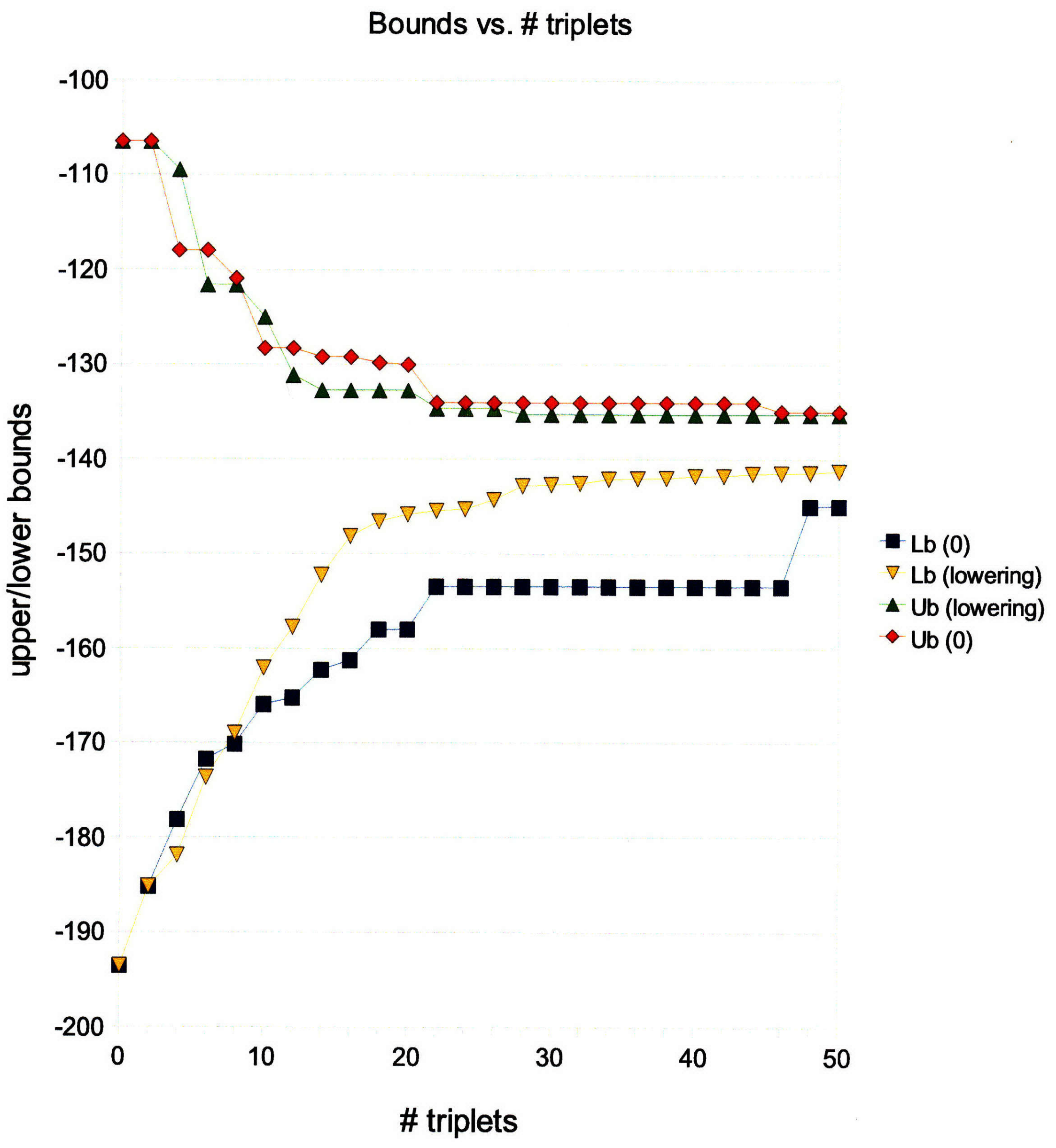
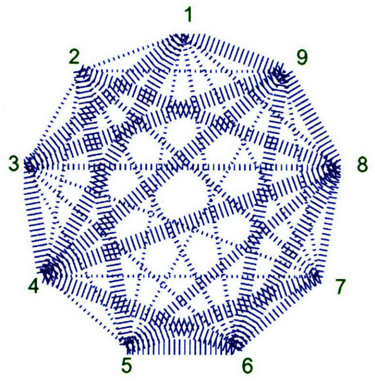
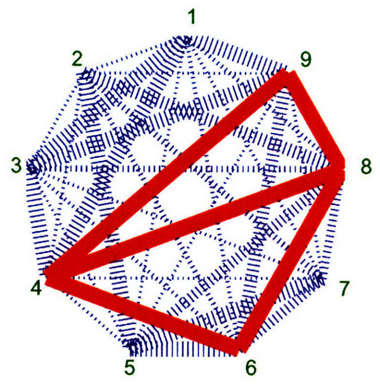


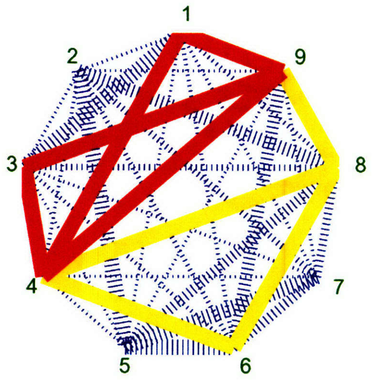
Figure 6-7: Upper and lower bounds from the subgradient method for different numbers of triplets selected by IQR scores for Case 44 of Table 5.1. The C_{IQR} lowering scheme is compared against the fixed $C_{IQR} = 0$ scheme. Bounds from the C_{IQR} lowering scheme approach the optimal value more quickly.



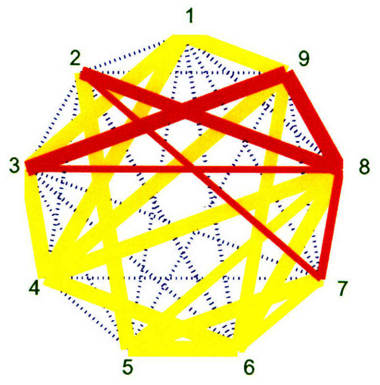
(a) No triplets.



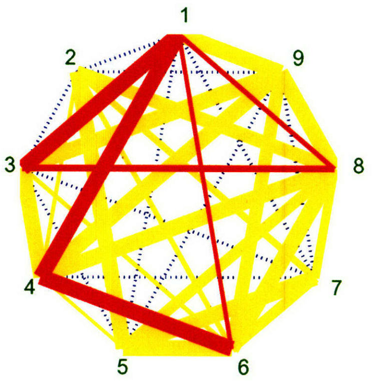
(b) Two triplets.



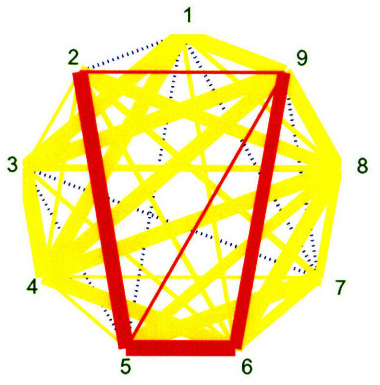
(c) Four triplets.



(d) 16 triplets.



(e) 32 triplets.



(f) 50 triplets.

Figure 6-8: Selection of triplets from adaptive lowering of C_{IQR} for Case 44 of Table 5.1. The test case preprocessed by DEE contains 9 residue positions, which are represented as vertices of a nonagon. The width of the dotted edge between a pair of positions correspond to the IQR value of the pairwise energies. Triplets from previous selections are shown in yellow, and those from the most recent selection are shown in red. Resulting lower bounds for each number of triplets included in the dual problem are shown in Figure 6-7.

3. done if the current number of triplets is greater than equal to N_{tri}.

4. identify n_{tri} new triplets with the highest disagreements based on the final solutions of subproblems obtained at the end of the subgradient method.

5. add the new triplets to the existing pool of triplets and newly construct a set of subgraphs by the procedure described in Section 6.6.2.

6. go back to step 2.

The result of applying the subgradient method with the above triplets selection scheme is disappointing as shown in Figure 6-9. The incremental addition of these triplets results in no more than a marginal improvement in the lower bounds. The upper bounds also show a very poor trend, which seems almost irrespective of the number of included triplets.

6.6.4 Incremental addition of triplets on the fly

As we have seen in Figure 6-6, the running time of the subgradient method almost linearly increases with the number of triplets included in the dual problem whereas the main improvement of the bounds are obtained by a relatively small number of triplets added early. This motivates a scheme that incrementally adds a small number of triplets while the subgradient method keeps running. The suggested procedure is as follows:

1. include an initial number of triplets in the dual problem.

2. apply the subgradient method for a constrained amount of time, such as for a fixed number of iterations, or until $\|\Delta\lambda\|_\infty < \epsilon'$ for some $\epsilon' > 0$.

3. done if the improvement of bound is not significant.

4. otherwise, identify next triplets and add to the dual problem.

5. go back to step 2.

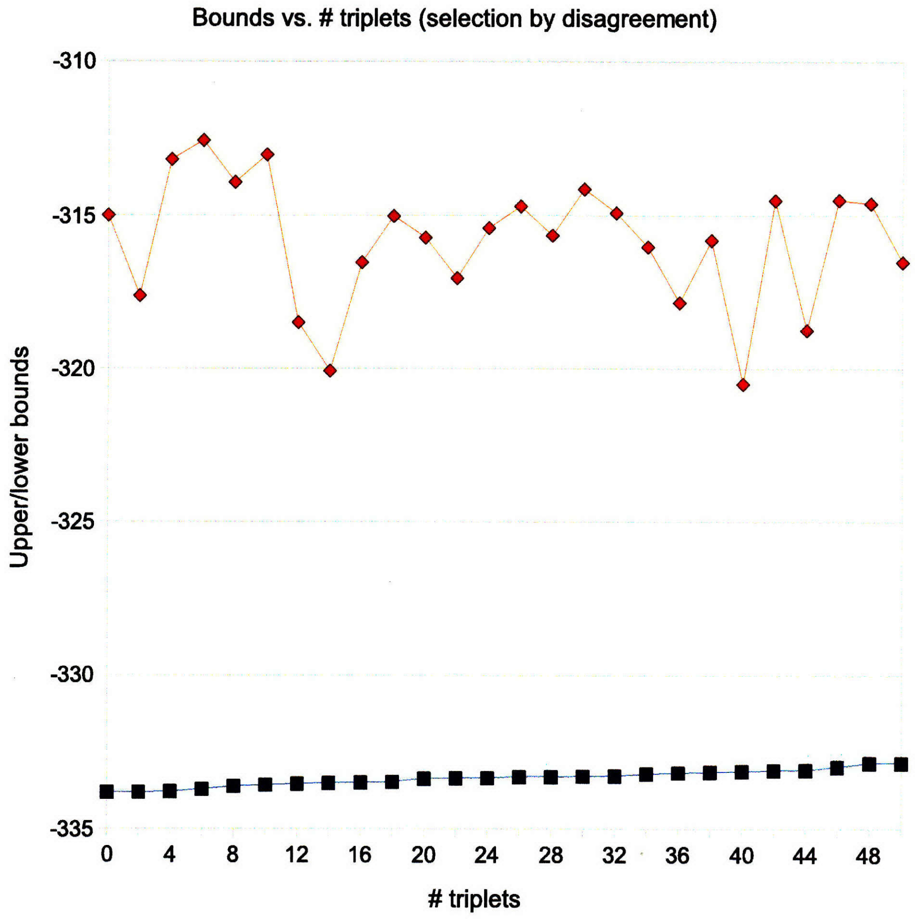


Figure 6-9: Bounds from adding triplets selected by disagreement scores.

Note that, when adding new triplets to the dual problem, the Lagrange multipliers for the existing triplets are kept as they are while those for the new triplets are all set to 0's. Such Lagrange multipliers will satisfy constraints 6.10 for the new dual problem. At the same time, they carry over the results from the previous subgradient method iterations in continuing to find the optimal solution for the refined dual problem, which saves unnecessary subgradient method iterations required if started fresh.

In our experiments, we tried using both fixed numbers of iterations and convergence constraints for step 2. Both attempts resulted in similar bounds in the end, but using fixed numbers of iterations was more straightforward in controlling the running times of the entire incremental scheme.

We applied the incremental addition of triplets to Case 44 of Table 5.1. Based on the previous observation that the running time of the subgradient method roughly grows linearly with the number of triplets, and that the improvement of bounds from addition of triplets decreases, we elaborated the above scheme with a heuristic way of determining the number of iterations for each number of triplets as follows:

1. set the number of iterations K_1 for the initial number of triplets ΔN_{tri}. Let $N_{curr} \leftarrow \Delta N_{tri}$ and $K \leftarrow K_1$.
2. run the subgradient method for K iterations.
3. done if $N_{curr} = N_{tri}$.
4. otherwise, add ΔN_{tri} triplets to the dual problem, and $N_{curr} \leftarrow N_{tri} + \Delta N_{tri}$, $K = K_1 \Delta N_{tri} / N_{curr}$.
5. go back to step 2.

We collected 50 triplets for the test case, and experimented with different ΔN_{tri} values; we let ΔN_{tri} to be 10, 25, or 50. To compare the running time, we made the total number of the subgradient method iterations equal for each ΔN_{tri} value. For example, we made the subgradient method run for 3,288 iterations for $\Delta N_{tri} = 50$,

Table 6.1: Bounds and running times of each batch of triplets addition and a fixed number of subgradient method iterations.

	$\Delta N_{tri} = 10$			$\Delta N_{tri} = 25$			$\Delta N_{tri} = 50$		
Batch	lb	ub	time	lb	ub	time	lb	ub	time
1	-166.86	-126.26	799	-145.77	-131.85	1,272	-141.57	-135.25	7,459
2	-159.57	-132.52	709	-143.62	-133.99	1,140			
3	-158.08	-132.52	672						
4	-157.47	-132.52	640						
5	-157.17	-132.52	651						
Total			3,471		2,412			7,459	

that is, $K_1 = 3,288$. For the run with $\Delta N_{tri} = 25$, we let $K_1 = 2,192$, which allows 1,096 subgradient method iterations after the second batch of 25 triplets are added. For the run with $\Delta N_{tri} = 10$, we let $K_1 = 1,440$.

Figure 6-10 shows the resulting lower bounds obtained from the incremental addition of triplets with different ΔN_{tri} values. Unlike previous figures, Figure 6-10 plots the bounds for each iteration of the subgradient method. $\Delta N_{tri} = 50$ corresponds to the case where we add all 50 available triplets to the dual problem from the start; therefore the incremental addition is not used. The plot shows that incremental addition of triplets result in weaker lower bounds than including all triplets from the start although the difference in the final lower bounds from $\Delta N_{tri} = 50$ and $\Delta N_{tri} = 25$ is relatively small. This suggests that using a fine grain of addition in the incremental scheme can be harmful in terms of the quality of lower bounds.

Table 6.1 summarizes the bounds and running times of each batch of triplets addition and a fixed number of subgradient method iterations. Apparently, using the incremental addition scheme reduced running times for $\Delta N_{tri} = 10$ and 25 whereas the strength of bounds were compromised. $\Delta N_{tri} = 25$ can be a reasonable trade-off between the running time and the strength of bounds. The fact that both the running time and bounds of $\Delta N_{tri} = 10$ are worse than those of $\Delta N_{tri} = 25$ suggests that the simple approximation of linear growth in running time with the number of triplets did not work well, and there is room for improvement of the incremental scheme for trade-off between running time and quality of bounds.

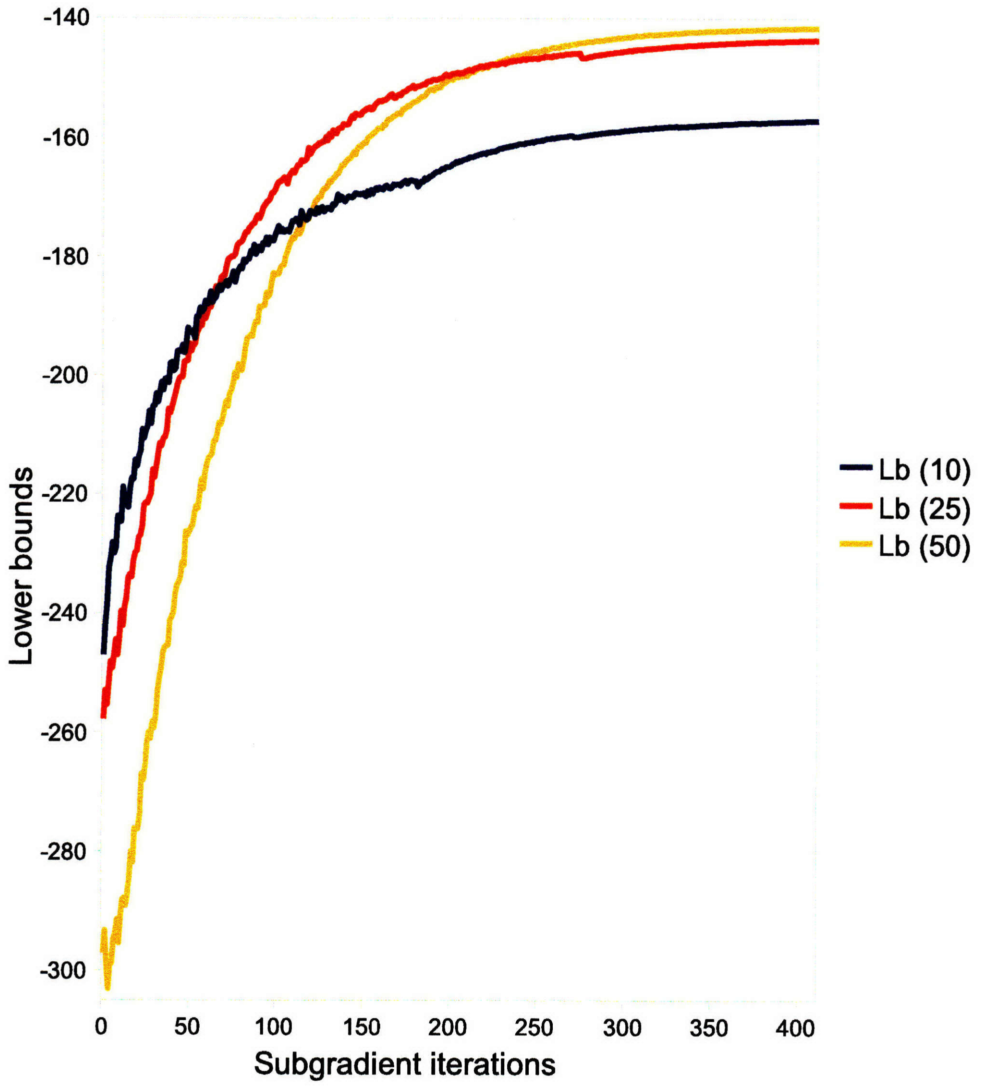


Figure 6-10: Lower bounds from different incrementally adding triplets. Each iteration shown on the x-axis corresponds to 8 actual iterations of the subgradient method. The legend shows the ΔN_{tri} value used to compute each curve of lower bounds.

6.7 Conclusions

We introduced the Lagrangian dual formulation of the GMEC problem. The formulation is based on decomposition with cyclic subgraphs. We provided an argument using the description of relaxed marginal polytopes that the optimal value of the dual problem can provide stronger lower bound for the GMEC problem than the tree-reweighted algorithms. The argument was verified by computational experiments, where dual problems derived from different numbers of triplets were solved by the subgradient method. We explored several heuristic approaches for selecting triplets. Triplets selected by IQR scores were seen to improve the bounds from the dual problem. When the dual problem derived only from trees and no triplets were solved by the subgradient method, it was found that the resulting lower bound is stronger than that from TRMP, but the running time is longer. We also explored a way of reducing the solution time of a dual problem derived from a large number of triplets by incrementally adding triplets on the fly while the subgradient method continues.

It is observed that the lower bound computed by the subgradient method tends to be stronger when using a small number of subgraphs than when spreading the triplets to many subgraphs. That is, for the same selection of triplets, and the same maximum treewidth constraint on subgraphs, the lower bound from a smaller number of subgraphs that packs them more effectively appears stronger than when constructing a set of subgraphs, each consisting of a single triplet and those edges uncovered by the set of triplets. This possibly suggests the additional cycles formed from neighboring triplets further constrains the dual relaxation. Another possible interpretation is that the subgradient method performs worse for a dual problem consisting of many small subproblems although the underlying theoretical lower bound should remain unaffected for different numbers of subproblems. This is a problem we would like to investigate further.

Chapter 7

Tightening the Lagrangian dual relaxation through addition of cuts

Finding good triplets and forming a Lagrangian dual relaxation based on them is a simple and effective way of obtaining strong lower bounds. However, aside from the technical issues on how to find a small number of such good triplets as discussed in Section 6.6, typically the following difficulties are encountered. Firstly, for small problems, there is only a limited number of triplets. Secondly, for dense problems, there may not be many triplets with a reasonable combined domain size because the number of rotamers of each position i, that is $|R_i|$, can be large. Solving a subproblem including such a triplet can be infeasible due to the system's memory constraint. Thirdly, because a subproblem should have a limited treewidth, the added triplets often need to be distributed over many subproblems. As a result, the number of subproblems to solve increases almost linearly, and so does the running time of one iteration of the subgradient method. Lastly, the effect of adding triplets on improving the lower bound quickly saturates after a small number of triplets. The significant increase in the running time with the number of added triplets can be too costly for only a small improvement in the lower bound.

In this chapter, we suggest a new method for tightening the lower bound by incrementally including valid constraints to the Lagrangian dual problem, and resolving the resulting dual problem, which is also known as "relax-and-cut" [Lucena, 2005].

7.1 Relax-and-cut

One systematic way of avoiding the problems with adding triplets while improving lower bounds from the Lagrangian dual relaxation is taking a more general perspective towards adding new constraints to our Lagrangian dual relaxation. This can be done by finding new inequalities, called cuts, that refine the relaxation we are optimizing over. Such an idea was previously explored in dynamically dualizing the constraints [Balas and Christofides, 1981, Hunting et al., 2001]. More recently, the idea was developed into a general framework called "relax-and-cut" [Escudero et al., 1994, Guignard, 1998, Lucena, 2005, Cavalcante et al., 2008].

Relax-and-cut in the context of the GMEC problem is easily understood by considering the linearized primal GMEC problem over the marginal polytope and the dual of the dual GMEC problem shown in Theorem 3. Figure 7-1 illustrates the inclusion relation between the marginal polytope $\mathcal{M}(\mathcal{G})$ and the polytope of problem (6.31), $\mathcal{P} = \bigcap_h \mathcal{M}(\mathcal{G};\mathcal{G}^h)$, that is, the intersection of marginal polytopes defined over subgraphs $\{\mathcal{G}^h\}$ and projected to $\mathcal{G}$. As Theorem 3 states the strong duality between the dual GMEC problem (6.12) and the linear program over $\mathcal{P}$, the optimal solution λ^* from the dual GMEC problem can be mapped to an extreme point $\hat{x}^1$ of $\mathcal{P}$[1]. However, in case $\hat{x}^1 \in \mathcal{P}\backslash\mathcal{M}(\mathcal{G})$, we attempt to exclude $\hat{x}^1$ from the primal problem in the hope of obtaining an optimal primal value or a tighter lower bound of the primal problem by re-solving the constrained problem. Such exclusion is done by finding a valid inequality $a \cdot x \leq \beta$ that is satisfied by all $\mu \in \mathcal{M}(\mathcal{G})$ but is not satisfied by x^*, and then adding the inequality to the description of $\mathcal{P}$. The problem of finding such (a, β) is called the separation problem [Nemhauser and Wolsey, 1988]. By Theorem 2, solving the primal linear program over the modified relaxation $\mathcal{P} \cap \{\tau \mid a \cdot \tau \leq \beta\}$ is equivalent in terms of the optimal value to solving the modified Lagrangian dual problem where the inequality is additionally dualized. Therefore, the modified Lagrangian dual problem is solved by the subgradient method, and the whole procedure is repeated until we do not find any more violated cuts.

[1]More precisely, there may exist more than one points on the face of $\mathcal{P}$ that have the same objective value as the dual objective value $q(\lambda^*)$

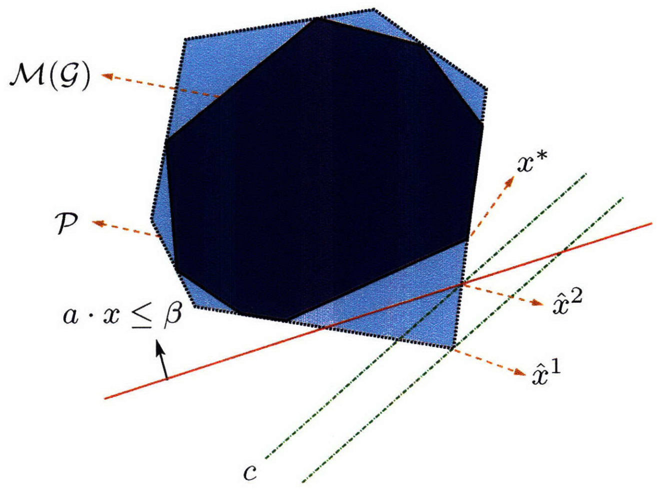


Figure 7-1: Tightening the relaxation by adding cuts in relax-and-cut. P is an outer relaxation of of $\mathcal{M}(\mathcal{G})$, $c \cdot x$ is the primal objective function, and x^* is the primal optimal solution. Initial solution of the Lagrangian dual relaxation finds the Lagrangian dual optimal λ^*, which has the same objective value as some $\hat{x}^1 \in P$. If we can compute $\hat{x}^1$ from λ^* and identify a valid facet-defining cut $a \cdot x \leq \beta$, adding the cut to the description of $\mathcal{P}$ can tighten the relaxation and may result in an improved lower bound. Note the extreme points of the marginal polytope all coincide with the boundary of the relaxation.

The description of the relax-and-cut method is very similar to that of the cutting plane method [Nemhauser and Wolsey, 1988]. The main difference is that the cutting plane method solves the primal problem and, therefore, has direct access to the primal solution whereas the relax-and-cut has to find a way of computing the primal solution from the dual solution so that it can generate valid cuts. Despite this advantage of the cutting plane method, that the primal solution is available to use, solving the primal linear program for the GMEC problem is often impractical because of the problem size and system constraints. Without the solution of the primal linear program, cuts cannot be found nor the basic LP bound obtained. On the other hand, relax-and-cut will only have approximate primal solutions, but relies on scalable dual solution methods and is more suitable for large GMEC problems. More general treatment and description of the cutting plane method, relax-and-cut, and another related method called "price-and-cut", can be found in Ralphs and Galati [2006].

7.2 Valid inequalities

It has been shown that a valid cut for the marginal polytope can be found by recognizing the equivalence of the binary marginal polytope and the cut polytope [Sontag and Jaakkola, 2007, Barahona and Mahjoub, 1986]. Other related works also include introducing triangle inequalities to represent a relaxation of the binary marginal polytope [Wainwright and Jordan, 2006, Globerson and Jaakkola, 2007b]. Partial constraint satisfaction problem for the minimum interference frequency assignment problem has the same polyhedral description as the GMEC problem, and its facet defining inequalities were investigated [Koster et al., 1998].

In this work, we use the approach suggested by Sontag and Jaakkola [2007], where the non-binary marginal polytope is mapped to a binary marginal polytope through a single projection, and the separation problem for the cut polytope is solved to find a violated valid cut for the marginal polytope, based on the equivalence of the binary marginal polytope and the cut polytope.

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a cut polytope $P_C(\mathcal{G})$ is the convex hull of incidence vectors of cuts of G, where

1. a cut $\delta(U)$ is the set of edges defined for given $U \subset \mathcal{V}$, such that exactly one end node is in U.

2. the incidence vector of x^C is defined for given cut C as

$$x^C(e) = \begin{cases} 1 & \text{if } e \in C, \\ 0 & \text{otherwise.} \end{cases} \tag{7.1}$$

If C is a simple cycle in $\mathcal{G}$, it is known that any $x \in P_C(\mathcal{G})$ satisfies the cycle inequality:

$$\sum_{e \in C \setminus F} x_e + \sum_{e \in F} (1 - x_e) \geq 1, \tag{7.2}$$

for any odd $F \subset C$, $|F|$.

A single projection π is specified by the set of partitions $\{\pi_i\}$, where

$$\pi_i : R_i \mapsto \{0, 1\}, \tag{7.3}$$

and $|\{r \in R_i \mid \pi_i(r) = 0\}| > 0$ and $|\{r \in R_i \mid \pi_i(r) = 1\}| > 0$ for all i. Then, the following linear map $\mathbb{R}^{d(G)} \mapsto \mathbb{R}^{2|\mathcal{E}|}$ based on the single projection can be defined [Sontag and Jaakkola, 2007]:

$$\begin{aligned} \tau^{\pi}_{ij}(x_i \neq x_j) &= \sum_{\{(r,s)\in R_i\times R_j \mid \pi_i(r)\neq\pi_j(s)\}} \tau_{i_r j_s} & (7.4)\\ \tau^{\pi}_{ij}(x_i = x_j) &= \sum_{\{(r,s)\in R_i\times R_j \mid \pi_i(r)=\pi_j(s)\}} \tau_{i_r j_s} & (7.5) \end{aligned}$$

Then, from the cycle inequality for the cut polytope, it can be shown that any $\tau \in \mathcal{M}(\mathcal{G})$ satisfies the following inequality [Sontag and Jaakkola, 2007]:

$$\sum_{(i,j)\in C\setminus F} \tau^{\pi}_{ij}(\pi_i \neq \pi_j) + \sum_{(i,j)\in F} \tau^{\pi}_{ij}(\pi_i = \pi_j) \geq 1, \tag{7.6}$$

for any cycle C in $\mathcal{G}$, and $F \subset C$, $|F|$ odd.

7.3 Solving the separation problem

The separation problem for the cycle inequality is solved by identifying a cycle in G that violates (7.6). Such a separation problem can also be generalized to consider more than one projections at a time by defining $k(i)$ projections $\{\pi^1_i, \pi^2_i, \ldots, \pi^{k(i)}_i\}$ for each position $i \in \mathcal{V}$, and defining a projection graph $\mathcal{G}_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ such that

$$\begin{aligned} \mathcal{V}_\pi &= \bigcup_{i\in\mathcal{V}} \{\pi^u_i \mid 1 \leq u \leq k(i)\} & (7.7)\\ \mathcal{E}_\pi &= \bigcup_{(i,j)\in\mathcal{E}} \{(\pi^u_i, \pi^v_j) \mid 1 \leq u \leq k(i),\ 1 \leq v \leq k(j)\}. & (7.8) \end{aligned}$$

Since there are no edges between projections from the same position, any cycle from $\mathcal{G}_\pi$ can be considered a cycle from a single projection graph.

Following is a description of the polynomial algorithm that solves the separation problem on $\mathcal{G}_\pi$ when attempting to separate τ:

1. Construct an undirected graph $\mathcal{G}'_\pi = (\mathcal{V}'_\pi, \mathcal{E}'_\pi)$ such that
   (a) $\mathcal{V}'_\pi$ consists of nodes $\pi_i^{u,1}$ and $\pi_i^{u,2}$ for each $\pi_i^u \in \mathcal{V}_{pi}$.
   (b) $\mathcal{E}'_\pi$ consists of edges $(\pi_i^{u,1}, \pi_j^{v,1})$, $(\pi_i^{u,1}, \pi_j^{v,2})$, $(\pi_i^{u,2}, \pi_j^{v,1})$, and $(\pi_i^{u,2}, \pi_j^{v,2})$ for each $(\pi_i^u, \pi_j^v) \in \mathcal{E}_\pi$.
   (c) edge weight $\tau_{ij}^\pi(x_i \neq x_j)$ is assigned to $(\pi_i^{u,1}, \pi_j^{v,1})$, $(\pi_i^{u,2}, \pi_j^{v,2})$.
   (d) edge weight $\tau_{ij}^\pi(x_i = x_j)$ is assigned to $(\pi_i^{u,2}, \pi_j^{v,1})$, and $(\pi_i^{u,2}, \pi_j^{v,2})$.
2. For each $\pi_i^u \in \mathcal{V}_\pi$, find the shortest path in $\mathcal{G}'_\pi$ from $\pi_i^{u,1}$ to $\pi_i^{u,2}$ with path length at least 3.

Any path from step 2 with cost less than 1 corresponds to a violated cut. The corresponding cycle can be easily retrieved by taking edges with different parities as belonging to F.

7.4 Estimating primal solutions

The premise for finding a violated cut is access to the optimal primal solution of the relaxed problem. This often becomes a technical challenge in using relax-and-cut because the dual solution does not directly translate to a primal solution. The problem of generating a primal solution has been investigated through extension of subgradient methods [Sherali and Choi, 1996, Barahona and Anbil, 2000].

In this work, we use simple algebra to estimate the primal solution from the decomposed parameters, the dual solution, and the optimal solutions for the subproblems. Let $\lambda^* \in \mathbb{R}^{\tilde{d}}$ the dual optimal solution for problem (6.12), $x^h(\lambda^{*,h}) \in \mathbb{R}^{d(\mathcal{G}^h)}$ an optimal solution[2] for the h^{th} subproblem given the Lagrangian multiplier λ^*, $c \in \mathbb{R}^{d(\mathcal{G})}$ the original parameter for the linearized primal problem, and $c^h \in \mathbb{R}^{d(\mathcal{G}^h)}$ the decomposed parameter for the hth subproblem. Then, approximate primal solution $\tilde{\mathbf{x}}$ can

[2]In case there are multiple optimal solutions, $x^h(\lambda^{*,h})$ is randomly selected.

be computed as a linear combination of subproblem solutions:

$$\tilde{x}_\alpha = \begin{cases} \sum_{h:\alpha\in\mathcal{I}(\mathcal{G}^h)} \frac{c_\alpha^h+\lambda_\alpha^{*,h}}{c_\alpha} x_\alpha^h & \text{if } c_\alpha \neq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{7.9}$$

for $\alpha \in \mathcal{I}(\mathcal{G})$, where the index set $\mathcal{I}(\mathcal{G})$ is defined as

$$\mathcal{I}(\mathcal{G}) = \left\{ \bigcup_{i\in\mathcal{V}} \{i_r \mid r \in R_i\} \right\} \cup \left\{ \bigcup_{(i,j)\in\mathcal{E}} \{i_r j_s \mid (r,s) \in R_i \times R_j\} \right\}. \tag{7.10}$$

The notations used in (7.9) are consistent with those used in Chapter 6; c_α is the linearized cost coefficient of the original problem for α^{th} component, $\lambda_\alpha^{*,h}$ is the dual optimal solution for h^{th} subproblem and α^{th} component, and x_α^h is α^{th} component of the h^{th} subproblem solution. Therefore, the approximate primal solution defined in (7.9) is a linear combination of subproblem solutions weighted by the ratio between the subproblem's cost coefficient penalized by the Lagrange multiplier and the original cost coefficient.

For such $\tilde{x}$, it is straightforward to see

$$c \cdot \tilde{x} = \sum_{\alpha\in\mathcal{I}(\mathcal{G})} c_\alpha \tilde{x}_\alpha = \sum_h \sum_{\alpha\in\mathcal{I}(\mathcal{G}^h)} (c_\alpha^h + \lambda_\alpha^{*,h}) x_\alpha^h = q(\lambda^*). \tag{7.11}$$

Despite its simple formula, $\tilde{x}$ obtained by (7.9) is not guaranteed to be primal feasible, or $\tilde{\mathbf{x}} \in \bigcap_h \mathcal{M}(\mathcal{G};\mathcal{G}^h)$. Even worse, it could be that $\tilde{x} \notin [0,1]^{d(\mathcal{G})}$. In practice, we add a positive offset to c_α so that $c_\alpha \geq 1$ for all $\alpha \in \mathcal{I}(\mathcal{G})$ in the hope of making $\tilde{x} \in [0,1]^{d(\mathcal{G})}$. We have also attempted to guide the update of λ so that $\tilde{x}$ always stays within $[0,1]^n$, but this severely degraded the quality of dual solution obtained. However, computation with actual test cases shows the pairwise components of $\tilde{x}$ are in $[0,1]^n$ in most cases (only the pairwise components are used to solve the separation problem for cycle inequalities).

Another problematic issue with estimating the primal solution from the dual solution is that the dual solution may not be dual optimal due to convergence problems of the subgradient method. In this case, the estimate can never be primal feasible because the dual objective value will be less than the primal optimal value.

Although the estimation may be far from being a perfect replacement of the optimal primal solution, it can still be useful in guiding the selection of cuts among the huge number of candidate cuts. It is also correct to add cuts found from using infeasible primal solution estimates because the class of cuts is valid for the marginal polytope. One thing that needs to be taken care of in using the estimate is that the weight assigned to edges of $\mathcal{G}'_\pi$ in solving the separation problem should not be negative in case a shortest path algorithm such as Dijkstra's algorithm is used.

7.5 Dualizing the cuts

The cuts found by solving the separation problem needs to be included in the existing Lagrangian dual relaxation. These cuts can be either added to the constraints of subproblems, or dualized by introducing new Lagrangian multipliers. In this work, we use the latter approach because this will allow us to use a consistent solution method for the subproblems.

Suppose we obtained W cycle inequalities from the previous dual solution:

$$-a^w \cdot x \leq -1, \quad w = 1, \ldots, W. \tag{7.12}$$

These are valid inequalities for the marginal polytope. Therefore, they can be added to the primal problem (6.4)–(6.5) without modifying its optimal solution. That is, the new unrelaxed primal problem using the linearized cost function is

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_h c^h \cdot x^h \\ \text{subject to} \quad & x \in \mathbb{R}^d, \ x^h \in \mathcal{F}^h, && (7.13) \\ & x^h = x_{|G^h} \ \forall h. && (7.14) \\ & -a^w \cdot x \leq -1, \quad w = 1, \ldots, W. && (7.15) \end{aligned}$$

We eventually want to get rid of x from the dual formulation. Therefore, constraints (7.15) are replaced with

$$-\sum_h a^{w,h} \cdot x^h \leq -1, \quad w = 1, \ldots, W., \tag{7.16}$$

where $a^{w,h} \in \mathbb{R}^{d(\mathcal{G}^h)}$, and

$$a_\alpha^w = \sum_{h:\alpha \in \mathcal{I}(\mathcal{G}^h)} a_\alpha^{w,h}, \tag{7.17}$$

for each $\alpha \in \mathcal{I}(\mathcal{G})$. Then, the Lagrangian function obtained by relaxing constraints (7.14) and (7.16) is

$$\begin{aligned}\hat{L}(\{x^h\}, x, \{\lambda^h\}, \{\xi^w\}) &= \sum_h c^h \cdot x^h + \sum_h \lambda^h \cdot (x^h - x_{|G^h}) - \sum_{w=1}^{W} \xi^w (\sum_h a^{w,h} \cdot x^h - 1) \\ &= \sum_h (c^h + \lambda^h - \sum_{w=1}^{W} \xi^w a^{w,h}) \cdot x^h - \sum_h \lambda^h \cdot x_{|G^h} + \sum_{w=1}^{W} \xi^w (7.18)\end{aligned}$$

By the same reasoning used in Section 6.2, the second term of (7.18) is dropped in the resulting dual function:

$$\hat{q}(\{\lambda^h\}, \{\xi^w\}) = \sum_h \min_{x^h \in \mathcal{F}^h} (c^h + \lambda^h - \sum_{w=1}^{W} \xi^w a^{w,h}) \cdot x^h + \sum_{w=1}^{W} \xi^w. \tag{7.19}$$

Finally, the dual problem with the dualized cuts is

$$\begin{aligned} \text{maximize} \quad & \hat{q}(\{\lambda^h\}, \{\xi^w\}) \\ \text{subject to} \quad & \lambda \in D, & (7.20) \\ & \xi^w \geq 0, \; w = 1, \ldots, W. & (7.21)\end{aligned}$$

Note that D, the domain of λ, is same as in the original dual problem before adding the cuts. In addition, the form of the subproblems was not changed except that the cost constants are adjusted by the new Lagrange multipliers $\{\xi^w\}$. Therefore, the same dynamic programming method can be used to solve the subproblems and compute subgradients. The subgradient method, however, now needs to update $\{\xi^w\}$ at each iteration. If we let

$$\hat{x}^h = \min_{x^h \in \mathcal{F}^h} (c^h + \lambda^h - \sum_{w=1}^{W} \xi^w a^{w,h}) \cdot x^h, \tag{7.22}$$

then, we have

$$\begin{aligned} \hat{q}(\{\lambda^h\}, \{\xi^w\}) &= \sum_h (c^h + \lambda^h - \sum_{w=1}^{W} \xi^w a^{w,h}) \cdot \hat{x}^h + \sum_{w=1}^{W} \xi^w \\ &= \sum_h c^h \cdot \hat{x}^h + \sum_h \lambda^h \cdot \hat{x}^h + \sum_{w=1}^{W} \xi^w \left\{ 1 - \sum_h a^{w,h} \cdot \hat{x}^h \right\}. \end{aligned} \tag{7.23}$$

This implies a subgradient for $\hat{q}(\cdot)$ at $\big\{\{\lambda^h\}, \{\xi^w\}\big\}$ can be described as

$$\left\{ \{\hat{x}^h\}, \left\{ 1 - \sum_h a^{w,h} \cdot \hat{x}^h \right\} \right\}, \tag{7.24}$$

as can be verified in a similar fashion to (6.36).

It should be already noted that, by adding cuts and avoiding the use of large triplets (or cliques in general), the number of variables in the dual problem increases. The increase in the number of dual variables adds computational cost because it implies a larger number of subgradient components to compute and Lagrange multipliers to be updated. In most subgradient method literature that uses the norm of the subgradient in computing the stepsize, the increase in the computational cost can be even larger, but we compute the stepsize by a simple geometric reduction rule.

In theory, the effect of including cliques in the dual problem can be simulated by using all necessary linear constraints that define the facets of the related marginal polytope [Wainwright and Jordan, 2006, Globerson and Jaakkola, 2007b]. Considering the large number of linear constraints required for such characterization of the polytope, the possible advantage of relax-and-cut over using all necessary triplets will come from finding as small a subset of the constraints as possible that are relevant to improving the dual optimal solution.

We use two schemes to maintain a reasonably small pool of cuts in relax-and-cut:

1. when adding a new cut to the pool (i.e. dual formulation), compare against all existing cuts to prevent duplication;

2. inactive cuts whose Lagrange multipliers are zero or very close to zero are deleted from the pool. This is done after adding new cuts so that we do not add back inactive cuts after deleting them.

More details on cut pool maintenance can be found in Lucena [2005].

7.6 Practical considerations

When the dual solution is already close to primal feasibility, addition of cuts to the dual problem would only modestly alter the dual solution. Therefore, when new cuts are identified and added to the dual problem, the values of Lagrange multipliers $\{\hat{\lambda}, \{\hat{\xi}^w\}\}$ from the previous subgradient method can be retained and used as the starting point of the next subgradient method in order to quickly converge to the next dual solution. However, if we also set $\xi^u = 0$ for all new cuts u, then the subsequent subgradient method will result in the same dual solution as the previous iteration since the $\{\hat{\lambda}, \{\hat{\xi}^w\}\}$ is where the previous subgradient method converged. Therefore, assigning a small positive number to the new Lagrange multipliers can be a good strategy to improve the dual solution and obtain quick convergence.

Additional speed up of relax-and-cut can be attained through a scheme called "non-delayed relax-and-cut" (NDRC) [Lucena, 2005], where new cuts are identified before the subgradient method converges. Such a scheme will compute an estimate of the primal solution, solve the separation problem with it, and add the resulting cuts every fixed number of iterations of the subgradient method. Considering the convergence of the subgradient method often takes very long, NDRC can possibly shorten the total running time although it may also generate many irrelevant cuts due to poor estimation of the primal solution.

Finally, the relax-and-cut algorithm used in this work is summarized by Algorithm 7.

Algorithm 7: Relax-and-cut algorithm.

```
   Data: Problem (7.21) with $W = 0$
   Result: Dual solution $\{\hat{\lambda}, \{\hat{\xi}^w\}\}$ for some $W \geq 0$
1  begin
2      Initialize $\lambda$, e.g. with 0's
3      repeat
4          Perform subgradient method until convergence by $\|\Delta\{\lambda, \{\xi^w\}\}\|_\infty < \epsilon'$
           or for a fixed number of iterations
5          Estimate a primal solution
6          Solve the separation problem
7          Add new cuts to the pool while checking duplication
8          Delete inactive cuts
9          Initialize $\xi^w \leftarrow \xi^{init} > 0$ for new cuts $w$
10         Initialize $\alpha \leftarrow \alpha^{init} > 0$
11     until no new cuts added
12 end
```

7.7 Computational results

We added the NDRC feature to the C++ implementation of the subgradient method from Chapter 6. The separation problem was solved by Yen's N-shortest paths algorithm [Martins and Pascoal, 2003]. The N-shortest paths algorithm was applied to find multiple paths from $\pi_i^{u,1}$ to $\pi_i^{u,2}$ for each $\pi_i^u \in \mathcal{V}_\pi$. The projection graph was constructed as a directed graph by allowing two edges in both directions for each pair of single projections derived from an edge in the original graphical model. We made N large enough so that we can obtain a desired number of cuts from one run of the N-shortest paths algorithm even if some of the paths founds already exist in the current cut pool.

We defined $k(i)$ single projections for each $i \in \mathcal{V}$, where $k(i)$ is the number of different rotamers choices in the subproblem solutions. Therefore, $k(i)$ can be no greater than $|\{h \mid i \in \mathcal{G}^h\}|$. We also gave a high priority to the single projection corresponding to rotamer i_r such that the approximate primal solution $\tilde{x}_{i_r}$ is fractional, that is, $|\tilde{x}_{i_r} - 0.5|$ is small. Single projections with high priorities will fill up the cut pool first when each single projection is allowed to generate a fixed number of cuts.

In our experiments with the NDRC algorithm, we regarded all cuts with Lagrange multipliers less than 10^{-3} as inactive and removed them from the cut pool at the end of a relax-and-cut iteration, that is, when the subgradient method roughly converges for the current set of cuts. We also need to determine ξ^{init}, the initial value of Lagrange multipliers for new cuts, and α^{init}, the initial stepsize to be used every time new cuts are added. Too large values for both parameters will result in long unnecessary fluctuations every time the subgradient method is started whereas too small values will result in little improvement in bounds. We experimented several values for ξ^{init} and α^{init}, together with values for ϵ' for the stopping condition $\|\Delta\{\lambda, \{\xi^w\}\}\|_\infty < \epsilon'$. As a result, the combination of $\xi^{init} = 0.005$, $\alpha^{init} = 0.005$, and $\epsilon' = 10^{-5}$ was found to work well in terms of running time and bounds improvement.

Our experiments using parameter values chosen above confirm that the NDRC algorithm certainly outperforms the delayed (ordinary) relax-and-cut (DRC) algorithm. The obvious advantage of NDRC over DRC is reduction of running time spent for making the subgradient method converge. On the other hand, the only problem with NDRC is that cuts are generated from approximate primal solutions, computed from suboptimal dual solutions obtained before the subgradient method converges. However, comparing the number of necessary cut generations and improvement of bounds from each relax-and-cut iteration of NDRC and DRC suggests that cuts generated from both approaches do not show any significant difference in tightening the relaxation. Such negligible difference in cuts generated by NDRC and DRC can be partially attributed to negligible difference in dual solutions used in computing approximate primal solutions because a large amount of computation in DRC is typically spent to obtain a very small improvement of the dual solution.

7.7.1 NDRC for different lengths of paths

We experimented with different cut lengths to observe the effect of cut lengths on the performance of NDRC. To do this, we only added violated valid cuts that are derived from cycles containing a specific number of edges. To add the same number of cuts with a specific length each time, the number of cycles to be found from the N-shortest paths algorithm was increased because many of N shortest paths may have different lengths. We experimented with cuts containing 3, 4, and 5 edges. In our experiments, it was found that cuts of longer lengths were harder to find. Due to relative scarcity of cuts with 4 or 5 edges, we added only 100 new cuts at each NDRC iteration.

Figure 7-2 shows the resulting NDRC bounds from different cut lengths for subproblem SWEET7-NOW-51-HIGH as the test case. The initial dual problem is derived from trees and 30 triplets selected by IQR scores. The plot clearly suggests that cuts of shorter lengths are more effective in tightening the relaxation. This is not very surprising considering the formulation of cuts (7.5)–(7.6), which probably provides more room for being satisfied when more number of edges, and therefore, variables are included. In other words, a constraint governing the choices of rotamers for many positions at a time can be relatively weak.

The number of active cuts in the cut pool also suggests different levels of cut relevance depending on the lengths of cuts. Figure 7-3 shows the number of active cuts at the beginning of each NDRC iteration. Although the same number of new cuts are added at each NDRC iteration, the slow increase in the number of active cuts for cut lengths 4 and 5 suggest that many of the added cuts turn out to be inactive, that is, insignificant, at the end of each NDRC iteration and deleted from the cut pool.

7.7.2 NDRC for different number of cuts added

In this section, we fix the length of cuts and explore the effect of adding different numbers of cuts. Adding more cuts will constrain the dual problem better, and will result in tighter bounds. However, we are interested in finding how such improved formulations are carried over to actual NDRC bounds and running times.

We applied NDRC to subproblem SWEET7-NOW-51-HIGH only using cuts derived from cycles with three edges. The initial dual problem was derived from trees and 30 triplets selected by IQR scores. For each independent run of NDRC, we varied

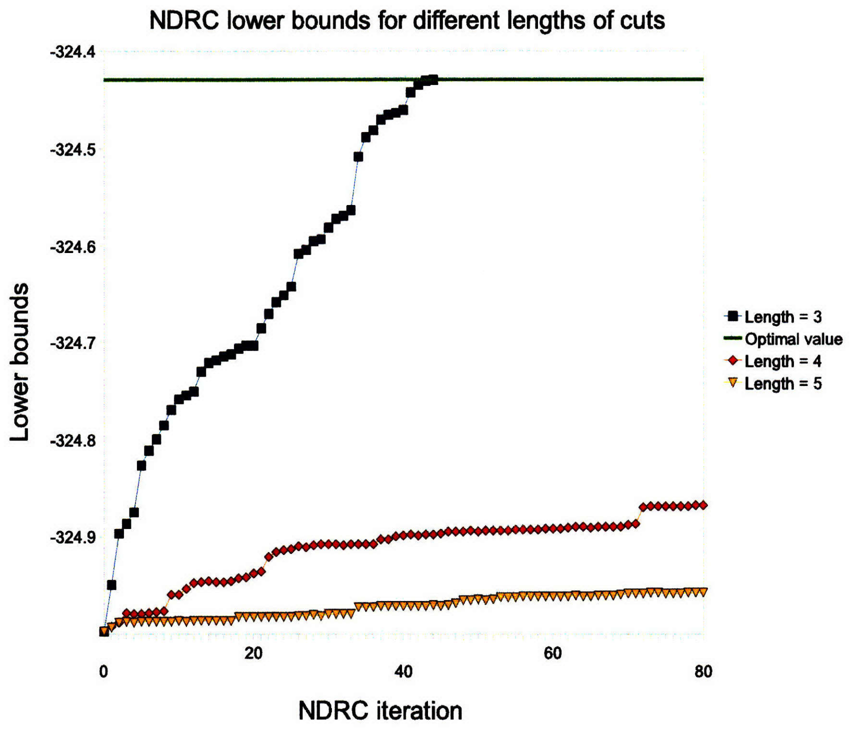


Figure 7-2: Change of NDRC lower bounds as cuts of different lengths are added at each NDRC iteration. Subproblem SWEET7-NOW-51-HIGH was used as the test case. One NDRC iteration corresponds to solution of the modified dual problem by the subgradient method, therefore hundreds of subgradient iterations. The lengths of cuts used to obtain each NDRC lower bounds curve are shown in the legend. NDRC with cut length 3 reaches the optimal value much faster than the other two.

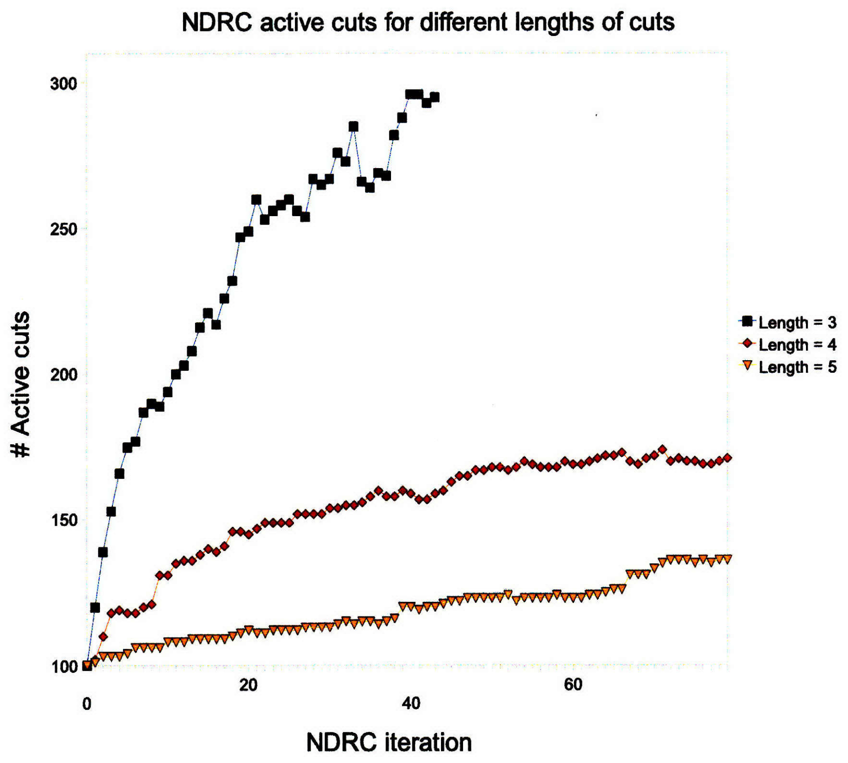


Figure 7-3: Change in the number of active cuts in the NDRC cut pool as cuts of different lengths are added at each NDRC iteration. Corresponding change in bounds are shown in Figure 7-2. The length of cuts used for each curve is shown in the legend. NDRC with cut length 3 retains more number of cuts added to the cut pool than the other two.

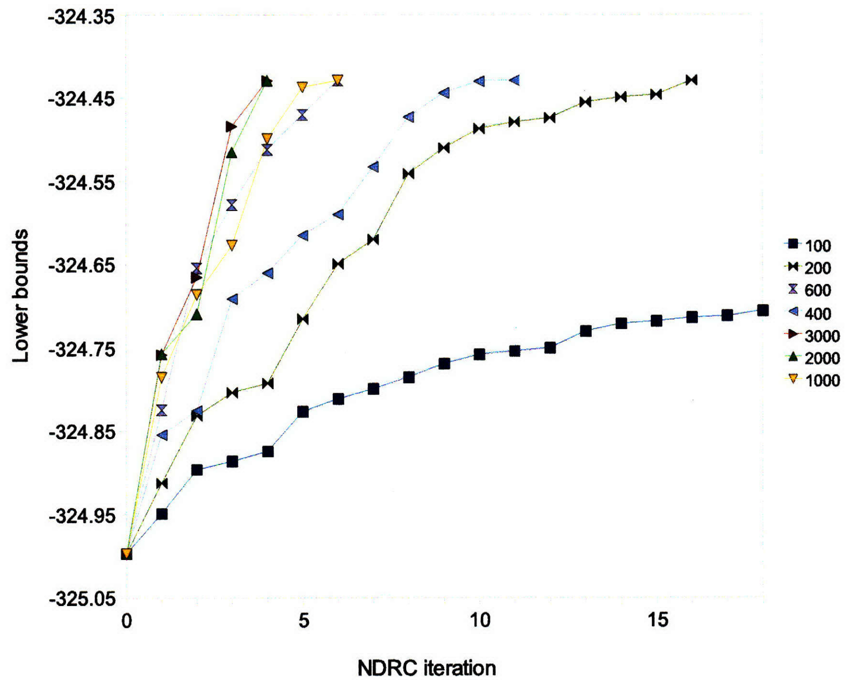


Figure 7-4: NDRC lower bounds from adding different numbers of cuts for subproblem SWEET7-NOW-51-HIGH. The legend shows the number of cuts to be added at each NDRC iteration. For the case '100', only the initial part up to iteration 18 is shown. The full curve for '100' can be found in Figure 7-2.

ΔN_{cut}, the number of cuts added at each NDRC iteration. We experimented with $\Delta N_{cut} = 100$, 200, 400, 600, 1000, 2000, and 3000. NDRC was run until the optimal value was found in each run. Note that a specific number of new cuts may not be found due to the lack of violated cuts with a specific length. This is more likely to happen when ΔN_{cut} is large. Therefore, the actual number of cuts added can be smaller than ΔN_{cut}.

Figure 7-4 shows the change of bounds for each ΔN_{cut} value. The lower bounds improve faster when ΔN_{cut} is larger. However, the difference in bounds among different ΔN_{cut} values decreases for large ΔN_{cut}. This is explained by Figure 7-5 to a certain extent. It shows that the difference in the numbers of active cuts for large ΔN_{cut} values is not as large as intended by ΔN_{cut} values.

Figure 7-6 shows cumulative running times of the subgradient method for each

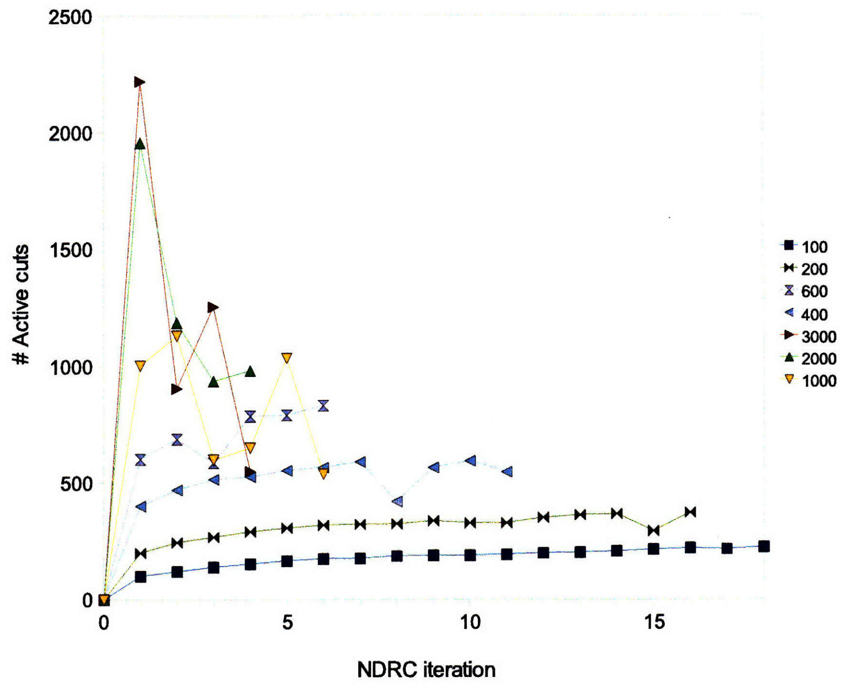


Figure 7-5: Numbers of active cuts for each NDRC iteration for the data shown in Figure 7-4.

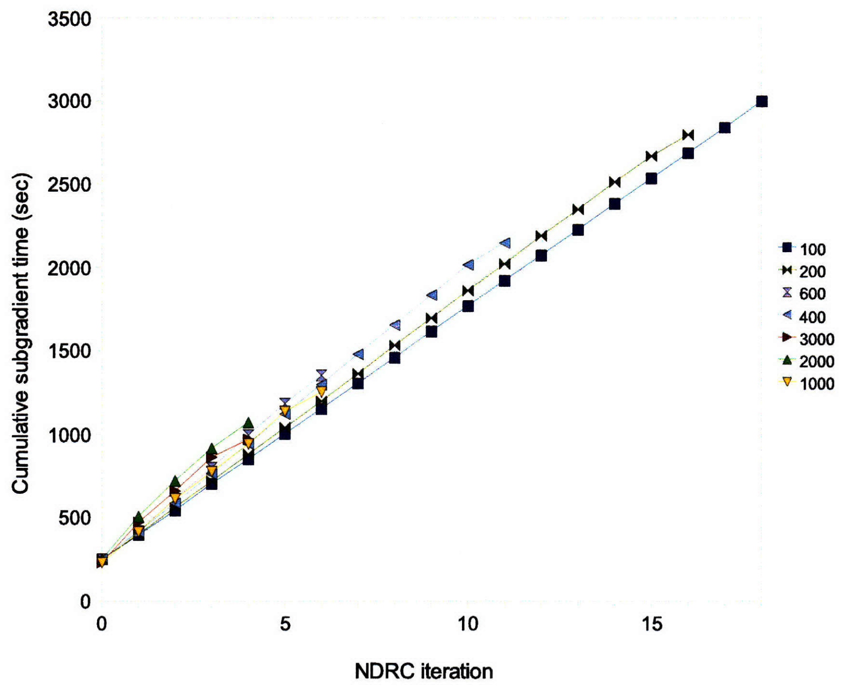


Figure 7-6: Cumulative running times of the subgradient method for the data shown in Figure 7-4.

ΔN_{cut}. The linear growth suggests that the running time of the subgradient method for each modified dual problem remains almost constant despite the increase in the number of Lagrange multipliers from addition of cuts. The small discrepancies in the rates of change among different ΔN_{cut} values reflect the differences in the number of cuts included in the dual problem. The total running times of the subgradient method for large ΔN_{cut} values tend to be small because fewer iterations are required for larger ΔN_{cut} values.

Finally, Figure 7-7 shows cumulative total running times of NDRC for each ΔN_{cut} value. Although not fully shown in the plot, $\Delta N_{cut} = 100$ has the longest running time of 11,300 seconds. The shortest total running time comes from $\Delta N_{cut} = 600$. The change in the trend shown from Figure 7-6 is due to the addition of time spent on solving the separation problem, that is, the cost of using the N-shortest paths algorithm. The running time of Yen's N-shortest path algorithm is $O(Nn(m+n\log n)$, where m, and n are the numbers of vertices and edges of the graph, respectively.

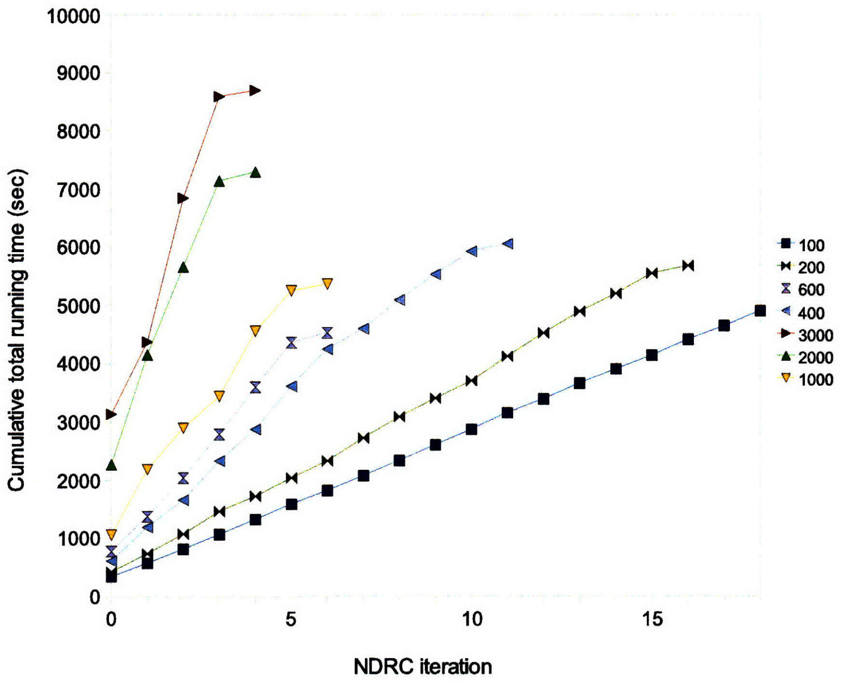


Figure 7-7: Cumulative total running times of NDRC including the subgradient method and the separation routine for the data shown in Figure 7-4.

Therefore, the separation routine takes longer for large ΔN_{cut} although it does not always find exactly ΔN_{cut} cuts, which was shown in Figure 7-5. Overall, despite the quick convergence of the subgradient method, the long separation times for large ΔN_{cut} values make them poor choices.

7.7.3 NDRC with random cuts

In this section, we evaluate the quality of approximate primal solutions, which are computed by (7.9) and used to generate cuts in our NDRC algorithm. Because the dual problem is modified by excluding the optimal solution of the current relaxation by adding valid inequalities, cuts generated for a non-optimal solution can be irrelevant to tightening the relaxation. The approximate primal solution computed in our NDRC algorithm has two possible problems: first, computation by (7.9) is not guaranteed to be correct, and second, the premature stopping of the subgradient method in NDRC may result in suboptimal dual solutions, and therefore even worse primal approximations. However, our experiments confirm that cuts generated from NDRC are equivalently effective as cuts generated by the ordinary relax-and-cut algorithm. Therefore, the problem with using suboptimal dual solutions is not considered significant.

One way of evaluating the approximate primal solutions is through comparison with exact primal solutions found by a linear programming solver. However, we do not have a complete LP formulation for the dual problem derived from multiple triplets. In addition, solving the LP for protein design cases is expensive. Therefore, we took an indirect approach of comparing cuts generated from the approximate primal solutions with randomly generated cuts.

We compared the performance of NDRC when 600 cuts of length 3 generated from approximate primal solutions are added and when 600 random cyclic inequalities (7.6) of length 3 are added. Figure 7-9 shows the resulting lower bounds for the two approaches applied to SWEET7-NOW-51-HIGH with the initial selection of 30 triplets. Note that the curve for random cuts is corresponds to average values from 10 independent runs. The plot clearly suggests that the cuts from approximate primal solutions lead to faster convergence to the optimal value.

Figure 7-9 shows the number of active cuts at the beginning of each NDRC iteration from the above simulation. The number of active cuts grows faster when approximate primal solutions are used except the downward spike at iteration 3. However, in the end, the final number of active cuts necessary to arrive at the optimal value with random cuts is larger. This suggests that the cuts from approximate primal solutions provide better guidance in finding the optimal value. The downward spike at iteration 3 implies a large number of cuts added at iteration 2 became satisfied, which could have led to a fast overall convergence of NDRC.

7.7.4 NDRC vs. addition of triplets

To compare the effect of adding triplets with adding cuts to the dual problem, we applied NDRC to dual problems for Case 44 of Table 5.1, derived from different

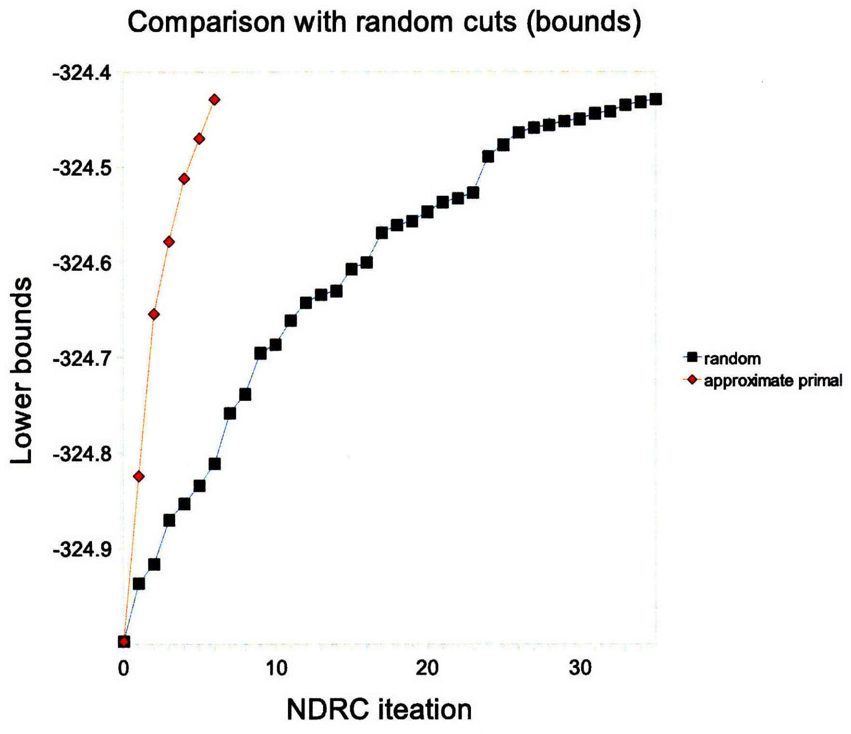


Figure 7-8: Bounds from adding cuts for test case 44 (EPO-INT-A-REG-11-11) of Table 5.1. Upper and lower bounds were calculated for different numbers of triplets selected by the magnitude of IQR and included in the relaxation. The numbers of triplets are indicated between parentheses in the legend. The optimal value (-135.9) of the problem is shown as the horizontal line.

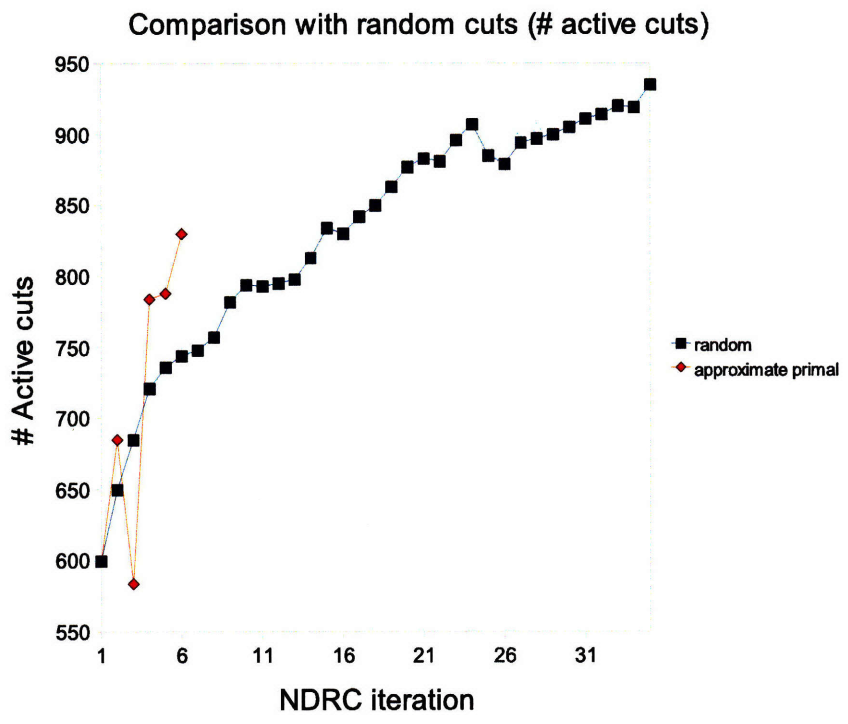


Figure 7-9: Bounds from adding cuts for test case 44 (EPO-INT-A-REG-11-11) of Table 5.1. Upper and lower bounds were calculated for different numbers of triplets selected by the magnitude of IQR and included in the relaxation. The numbers of triplets are indicated between parentheses in the legend. The optimal value (-135.9) of the problem is shown as the horizontal line.

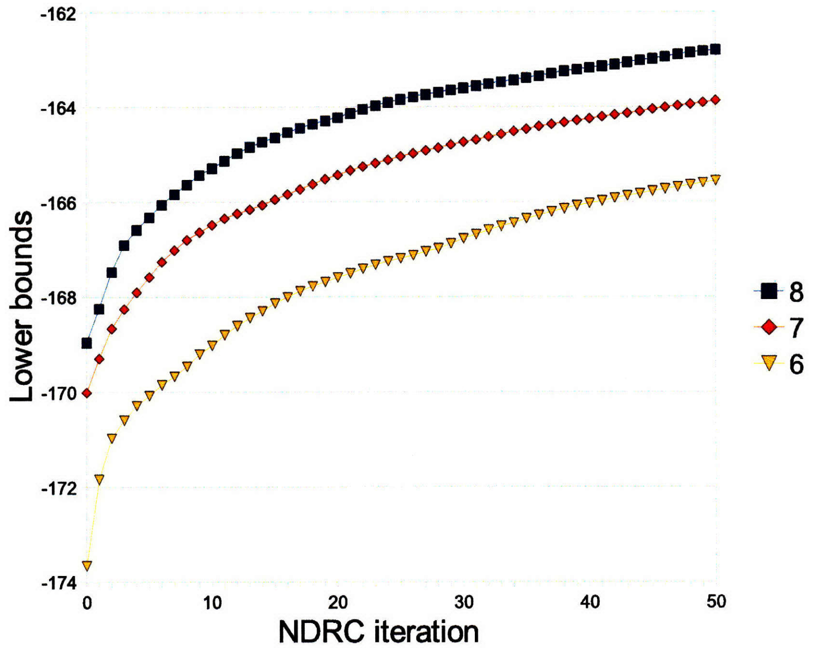
numbers of triplets. NDRC generated at most 1,000 new cuts constructed from cycles with three edges at each NDRC iteration. For each independent run of NDRC, we fixed the number of triplets to 6, 7, or 8 and repeated 50 NDRC iterations. The triplets used for each run are maximally overlapped; for example, the 8 triplets include all 6 triplets used to construct the smallest dual problem.

Figure 7-10 shows the results. The figure at the top plots the resulting lower bounds for each NDRC iteration and number of triplets. As expected, the lower bound is tighter for larger number of triplets at NDRC iteration 0, that is, when the number of cuts is 0. It is also observed that the bounds curve do not cross one another at any iteration. However, the difference in bounds from different numbers of triplets are overcome to a certain degree through addition of cuts. For example, at iteration 5, the lower bound from 6 triplets becomes almost equal to the initial lower bound from 7 triplets.

The figure at the bottom of Figure 7-10 shows cumulative running times of NDRC at each NDRC iteration. The rate of increase in the cumulative running time is larger when more triplets are included because the subgradient method for a larger dual problem takes longer. The plot suggests the cost of achieving the improvement of bounds through addition of cuts is quite large. For example, the initial lower bound with 7 triplets is computed in 734 seconds whereas the equivalent lower bound from 6 triplets at iteration 5 is computed in 3,590 seconds. Figure 7-11 plots the data shown in Figure 7-10 again to visualize the difference in NDRC running times between different numbers of triplets when similar bounds are to be obtained.

7.8 Conclusions

In this chapter, we described a non-delayed relax-and-cut (NDRC) algorithm that identifies relevant cuts from a dual solution and computes stronger lower bounds by solving the dual problem modified through addition of the cuts. We studied various components of the method that may affect the resulting lower bounds and the running time, such as the length of a cycle that a cut is constructed from, the number of cuts



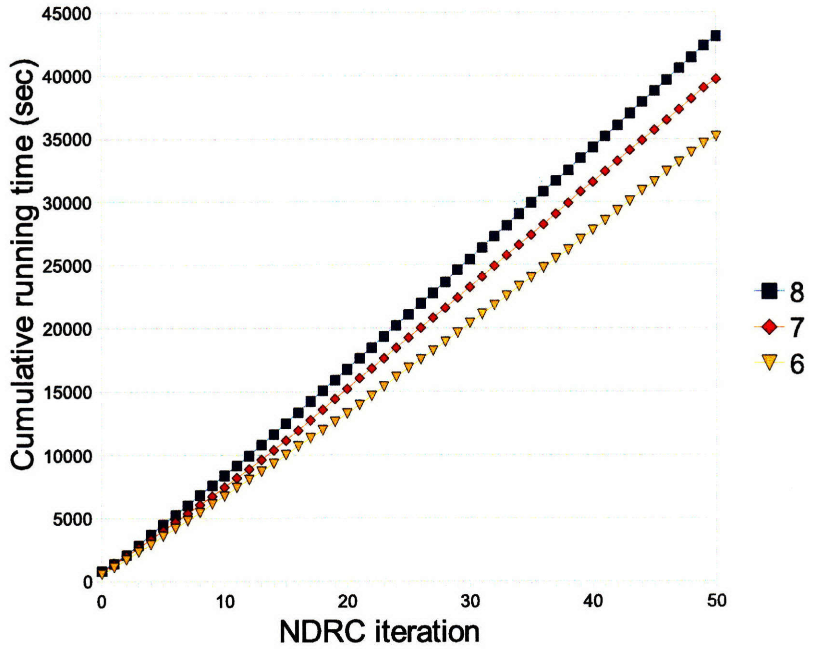


Figure 7-10: NDRC was applied to dual problems for Case 44 of Table 5.1, derived from different numbers of triplets: (top) lower bounds computed by NDRC, (bottom) cumulative running time of NDRC. Numbers in the legend indicate numbers of triplets.

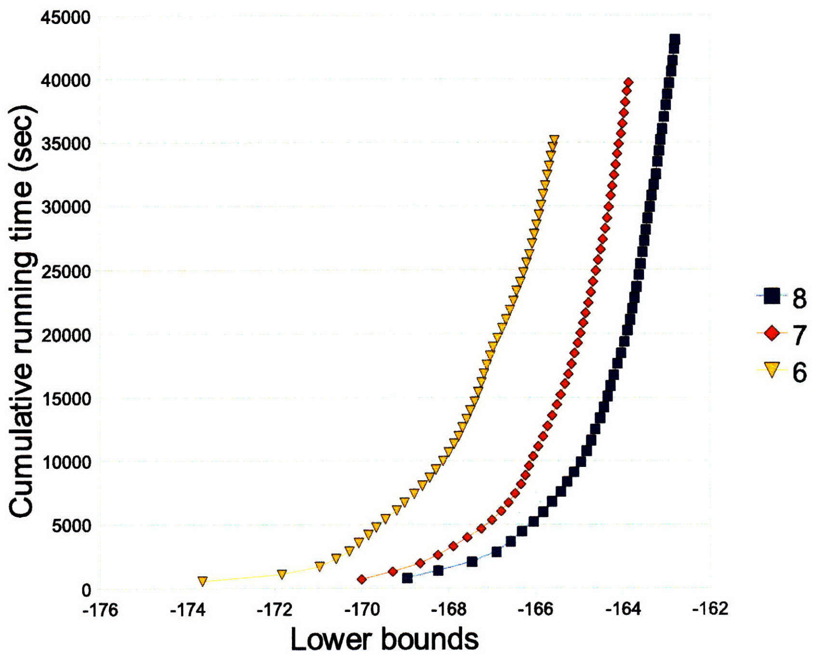


Figure 7-11: Plot of NDRC lower bound vs. cumulative NDRC running time for data shown in Figure 7-10.

added at each NDRC iteration, relevance of cuts generated from approximate primal solutions compared to random cuts. We also compared the bounds and the running time when dual problems are derived from different numbers of cuts and triplets.

We found that using additional triplets is more effective in terms of both lower bound and running time than repeating NDRC iterations. However, when the number of triplet configurations is too large for the system memory, NDRC can be an alternative method for obtaining incremental lower-bound improvement because adding cuts only involve linear growth of memory usage. It is also observed that the effect of adding cuts tends to decrease with NDRC iterations. Therefore, using a few iterations of NDRC as a post-process of solving the initial dual problem can be a good strategy.

As a future work, the following extensions of NDRC is suggested:

1. development of a different scheme to compute more reasonable approximate primal solutions from dual solutions;

2. optimization of the separation routine to solve the separation problem more

efficiently and generate more number of unique short cycles;

3. use of different classes of cuts than cyclic inequalities that might constrain the relaxation better.

Chapter 8

Individual lower bounds from Lagrangian dual relaxation and their uses in BroMAP

In Chapter 6 and 7, we have seen how we can obtain strong lower bounds by including cliques of variables and violated valid cuts. Although we can achieve incremental improvements of bounds through these procedures, the computational cost can be at times too expensive compared to the gain in tightness of bounds. In the case of including variable cliques, the cardinality of some clique domain size can be prohibitively large, leading to memory exhaustion or a long running time for each subgradient method iteration. Adding cuts has much less effect on the time taken for subgradient method iterations, but the gain in the tightness of bounds are relatively small compared to adding triplets. Therefore, relax-and-cut will require many iterations of solving the separation problem and optimizing a further constrained relaxation problem. For large primal problems, such iterations of relax-and-cut can be expensive.

In this chapter, we look into a way of computing lower bounds for individual rotamers and rotamer pairs from the dual formulation. The idea is that these bounds can be tight enough to eliminate some rotamers or rotamer pairs from the problem even though the global lower bound for a problem might be too weak to find an optimal solution of prune the entire subproblem. When it is difficult to include another variable clique or relax-and-cut is expensive for the gain in tightness of bounds, it can be useful to shed unnecessary rotamers and rotamer pairs first and proceed to the next step of refining the bounds. This approach is similar to the techniques used in BroMAP of Chapter 3, where we were able to find a simple rule to compute rotamer and rotamer-pair lower bounds using pseudo-max-marginals from the tree-reweighted max-product algorithm.

The main differences between TRMP and the dual formulation is that we do not obtain pseudo-max-marginals for the dual formulation at the end of the subgradient method, and that we need to deal with non-tree structures of subproblems. To obtain the bounds without pseudo-max-marginals in the dual framework, we are going to compute max-marginals for each subproblem through the junction tree algorithm and combine them to compute rotamer and rotamer-pair lower bounds for the original problem.

In Section 8.1, we will review the junction tree algorithm. Section 8.2 describes a way of computing rotamer and rotamer pair lower bounds from the dual formulation. Section 8.3 provides computational results on the tightness of these bounds and their computation time. We also provide some results of including these bounds in BroMAP and applying it to a few hard test cases of Table 5.1.

8.1 Junction tree algorithm

The junction tree algorithm is an extension of the max-product algorithm to junction trees instead of simple trees [Cowell et al., 1999]. Junction tree is simply a different name of tree decomposition of Section 6.5.1. Therefore, the junction tree algorithm can be regarded as a different form of dynamic programming that can be used to find the MAP assignment on cyclic graphical models. The main difference between the dynamic programming algorithm presented in Section 6.5.2 and the junction tree algorithm is that the junction tree algorithm provides the max-marginals table, that is, the optimal objective value of the problem when each variable clique is fixed to each possible assignment value. On the other hand, the outcome of the dynamic programming will be only useful for finding an optimal assignment when the assignment for the root node is fixed. When we want to find the optimal value or assignment when a node other than the root is fixed to a certain assignment, we will need to rerun the dynamic programming algorithm with the new node designated as the root. This relation between the junction tree algorithm and dynamic programming algorithm is analogous to that between the max-product algorithm and the dynamic programming on trees. In summary, the junction tree algorithm is a more convenient tool than the dynamic programming when computing individual lower bounds by providing the max-marginal table for each variable clique.

Algorithm 8, 9, and 10 describe the junction tree algorithm for the min-sum computation. Although Cowell et al. [1999] describes the sum-product version of the junction tree algorithm, the results on the sum-product junction tree algorithm are transferable to the mim-sum version due to the applicability of the "general distributive law" to the general semiring case [Aji and McEliece, 2000]. Note that Algorithm 8, 9, and 10 is called Hugin architecture. There also exist other forms of the junction tree algorithm such as Lauritzen-Spiegelhalter (LS) architecture, which is very similar to Hugin architecture, and a message-passing algorithm, called Shenoy-Shafer (SS) architecture [Lepar and Sehnoy, 1998]. Hugin architecture requires more memory than LS architecture because of additional storage for separator messages, but Hugin requires less computation than LS. It is claimed SS is computationally more efficient than Hugin on average and also provide single marginals at the end of propagation [Lepar and Sehnoy, 1998]. We use Hugin architecture in this work mainly for its simple form of presentation and ease of implementation.

The key properties of the Hugin propagation can be summarized as follows:

1. The equality $f(\mathbf{x}) = \sum_{u \in I} \phi^u(\mathbf{x}^u) - \sum_{(u,v) \in F} \lambda^{u \cap v}(\mathbf{x}^{u \cap v})$ holds for all $\mathbf{x}$ after each transfer of separator and node messages, that is, either after performing steps 4, 6, and 7 of JT-Bottom-To-Top or steps 3, 4, and 5 of JT-Top-To-Bottom.

2. Any time after both JT-Bottom-to-Top and JT-Top-To-Bottom messages have passed between two nodes $u, v \in I$, $(u, v) \in F$, the node messages and the

Algorithm 8: Junction tree algorithm for the min-sum computation. The minimum value of $f(\mathbf{x})$ for each assignment of $\mathbf{x}^u$ and $\mathbf{x}^{pa(u)\cap u}$ is computed and output as $\phi^u(\mathbf{x}^u)$ and $\lambda^{pa(u)\cap u}(\mathbf{x}^{pa(u)\cap u})$, respectively.

```
Data: $f(\mathbf{x})$, $(\{X_i \mid i \in I\}, T = (I, F))$ as in Algorithm 4.
Result: $\phi^u(\mathbf{x}^u)$ for each $u \in I$, and $\lambda^{u\cap v}(\mathbf{x}^{u\cap v})$ for each $(u,v) \in F$.
1 begin
2     Initialize $\{\phi^u, \lambda^{u\cap v}\}$ so that $f(\mathbf{x}) = \sum_{u\in I} \phi^u(\mathbf{x}^u) - \sum_{(u,v)\in F} \lambda^{u\cap v}(\mathbf{x}^{u\cap v})$, e.g.
      by $\phi^u(\mathbf{x}^u) = f(\mathbf{x}^u)$ for $u \in I$, and $\lambda^{u\cap v}(\mathbf{x}^{u\cap v}) = 0$ for $(u,v) \in F$.
3     JT-Bottom-to-Top($\Upsilon$, $\phi^\Upsilon$, $\{\lambda^{\Upsilon\cap i}, i \in D(\Upsilon)\}$, $\{\lambda_{old}^{\Upsilon\cap i}, i \in D(\Upsilon)\}$)
4     JT-Top-to-Bottom($\Upsilon$, $\phi^\Upsilon$, {}, {})
5 end
```

Algorithm 9: JT-Bottom-to-Top procedure of the junction tree algorithm. Node message ϕ^u is updated based on the separator messages $\lambda^{u\cap i}$ passed from descendents $i \in D(u)$. Then, the current separator message of $pa(u)\cap u$, that is, $\lambda^{pa(u)\cap u}$ is copied to $\lambda_{old}^{pa(u)\cap u}$, and is updated based on the updated node message ϕ^u.

```
Data: $u \in I$, $\phi^u$, $\{\lambda^{u\cap i}, i \in D(u)\}$, $\{\lambda_{old}^{u\cap i}, i \in D(u)\}$
Result: $\phi^u$, $\lambda^{pa(u)\cap u}$, $\lambda_{old}^{pa(u)\cap u}$
1 begin
2     foreach $i \in D(u)$ do
3         JT-Bottom-to-Top($i$, $\phi^u$, $\{\lambda^{u\cap i}, i \in D(u)\}$, $\{\lambda_{old}^{u\cap i}, i \in D(u)\}$)
4         $\phi^u(\mathbf{x}^u) \leftarrow \phi^u(\mathbf{x}^u) + \lambda^{u\cap i}(\mathbf{x}^{u\cap i}) - \lambda_{old}^{u\cap i}(\mathbf{x}^{u\cap i})$.
5     if $u \neq \Upsilon$ then
6         $\lambda_{old}^{pa(u)\cap u}(\mathbf{x}^{pa(u)\cap u}) \leftarrow \lambda^{pa(u)\cap u}(\mathbf{x}^{pa(u)\cap u})$
7         $\lambda^{pa(u)\cap u}(\mathbf{x}^{pa(u)\cap u}) \leftarrow \min_{\{\tilde{\mathbf{x}}^u \mid \tilde{\mathbf{x}}^{pa(u)\cap u} = \mathbf{x}^{pa(u)\cap u}\}} \phi^u(\tilde{\mathbf{x}}^u)$
8 end
```

Algorithm 10: JT-Top-to-Bottom procedure of the junction tree algorithm. The same operations as in JT-Bottom-To-Top procedure are performed in the reverse direction.

```
  Data: $u \in I$, $\phi^u$, $\lambda^{pa(u)\cap u}$, $\lambda_{old}^{pa(u)\cap u}$
  Result: $\phi^u$, $\lambda^{pa(u)\cap u}$, $\lambda_{old}^{pa(u)\cap u}$
1 begin
2     if $u \neq \Upsilon$ then
3         $\lambda_{old}^{pa(u)\cap u}(\mathbf{x}^{pa(u)\cap u}) \leftarrow \lambda^{pa(u)\cap u}(\mathbf{x}^{pa(u)\cap u})$
4         $\lambda^{pa(u)\cap u}(\mathbf{x}^{pa(u)\cap u}) \leftarrow \min_{\{\tilde{\mathbf{x}}^{pa(u)} \mid \tilde{\mathbf{x}}^{pa(u)\cap u}=\mathbf{x}^{pa(u)\cap u}\}} \phi^{pa(u)}(\tilde{\mathbf{x}}^{pa(u)})$
5         $\phi^u(\mathbf{x}^u) \leftarrow \phi^u(\mathbf{x}^u) + \lambda^{pa(u)\cap u}(\mathbf{x}^{pa(u)\cap u}) - \lambda_{old}^{pa(u)\cap u}(\mathbf{x}^{pa(u)\cap u})$.
6     foreach $i \in D(u)$ do
7         JT-Top-to-Bottom($i$, $\phi^u$, $\lambda^{pa(u)\cap u}$, $\lambda_{old}^{pa(u)\cap u}$)
8 end
```

separator message are consistent in that

$$\min_{\{\tilde{\mathbf{x}}^u \mid \tilde{\mathbf{x}}^{u\cap v}=\mathbf{x}^{u\cap v}\}} \phi^u(\tilde{\mathbf{x}}^u) = \lambda^{u\cap v}(\mathbf{x}^{u\cap v}) = \min_{\{\tilde{\mathbf{x}}^v \mid \tilde{\mathbf{x}}^{u\cap v}=\mathbf{x}^{u\cap v}\}} \phi^v(\mathbf{x}^v) \tag{8.1}$$

3. After completion of the Hugin propagation, both node and separator messages associated each node are "min-sums", that is,

$$\phi^u(\mathbf{x}^u) = \min_{\{\tilde{\mathbf{x}} \mid \tilde{\mathbf{x}}^u=\mathbf{x}^u\}} f(\tilde{\mathbf{x}}) \ \ \forall u \in I, \tag{8.2}$$

$$\lambda^{u\cap v}(\mathbf{x}^{u\cap v}) = \min_{\{\tilde{\mathbf{x}} \mid \tilde{\mathbf{x}}^{u\cap v}=\mathbf{x}^{u\cap v}\}} f(\tilde{\mathbf{x}}) \ \ \forall (u,v) \in F \tag{8.3}$$

Note that min-sums of (8.2) and (8.3) are analogous to max-marginals of the max-product algorithm. However, unlike the definition of max-marginals, these equations are satisfied without the need of any constant factors.

Min-sums of a variable or a pair of variables can be computed by enumerating all assignment values in a variable clique. For example, if $i, j \in X_u$, $k \in X_u \cap X_v$, $u, v \in I$, the min-sums $\min_{\{\tilde{\mathbf{x}} \mid (\tilde{x}_i,\tilde{x}_j)=(x_i,x_j)\}} f(\mathbf{x})$ and $\min_{\{\tilde{\mathbf{x}} \mid \tilde{x}_k=x_k\}} f(\mathbf{x})$ can be computed as

$$\min_{\{\tilde{\mathbf{x}} \mid (\tilde{x}_i,\tilde{x}_j)=(x_i,x_j)\}} f(\mathbf{x}) = \min_{\{\tilde{\mathbf{x}}^u \mid (\tilde{x}_i,\tilde{x}_j)=(x_i,x_j)\}} \phi^u(\mathbf{x}^u), \tag{8.4}$$

$$\min_{\{\tilde{\mathbf{x}} \mid \tilde{x}_k=x_k\}} f(\mathbf{x}) = \min_{\{\tilde{\mathbf{x}}^{u\cap v} \mid \tilde{x}_k=x_k\}} \lambda^{u\cap v}(\mathbf{x}^{u\cap v}). \tag{8.5}$$

One strategy to reduce the amount of enumeration to compute min-sums for a pair of variables is finding a node or a separator with the minimum cardinality that contains both variables. Once all pairwise min-sums are computed, the singleton min-sums can be readily derived from associated pairwise min-sums.

8.2 Computing individual bounds from the dual formulation

Once we compute all singleton and pairwise min-sums for each subproblem of the Lagrangian dual formulation, lower bounds for each rotamer and rotamer pair can be easily computed. Suppose we want to compute the minimum value of (6.1) when the assignment of rotamers for pairs of variables $(\eta, \zeta) \in \mathcal{E}$ is fixed to some $(r, s) \in R_\zeta \times R_\eta$, that is,

$$\begin{array}{ll} \text{minimize} & \sum_{i \in V} \sum_{r \in R_i} E(i_r) x_{i_r} + \sum_{(i,j) \in E} \sum_{(r,s) \in R_i \times R_j} E(i_r j_s) x_{i_r j_s} \\ \text{subject to} & x \in \mathcal{F},\ x_{\eta_r \zeta_s} = 1. \end{array} \tag{8.6}$$

Finding the exact minimum value for the modified primal problem is still difficult, but we can obtain a lower bound of the minimum value through the same Lagrangian dual relaxation described in Section 6.2. Through an identical derivation, the resulting dual problem corresponding to (6.12) is

$$\begin{array}{ll} \text{maximize} & \sum_{h:(\eta,\zeta) \in \mathcal{E}^h} \min_{\{x^h \in \mathcal{F}^h | x_{\zeta_r \eta_s} = 1\}} (c^h + \lambda^h) \cdot x^h \\ & \quad + \sum_{h:(\eta,\zeta) \notin \mathcal{E}^h} \min_{x^h \in \mathcal{F}^h} (c^h + \lambda^h) \cdot x^h \\ \text{subject to} & \lambda \in D. \end{array} \tag{8.7}$$

Weak duality states that the objective function of (8.7) is a lower bound of (8.6) for all $\lambda \in D$. Therefore, the rotamer lower bound for (η_r, ζ_s) can be computed by solving the modified subproblems with values of λ from the convergence of the subgradient method, and then summing the resulting objective values from each subproblem.

Note that the second summation of the objective function of (8.7) corresponds to the sum of optimal values of unmodified subproblems, which are trivially obtained from the solution of each subproblem.

In general, we prefer having rotamer-pair lower bounds at least for all rotamer pairs connected by an edge in the original graphical model. The procedure for computing a rotamer-pair lower bound described above is straightforward but repeating the procedure as many times as the number of rotamer pairs in the problem can be prohibitive for large problems. However, as in individual bounds computation from TRMP, the computational burden can be relieved by using min-sums from the junction tree algorithm where

$$\min_{\{x^h \in \mathcal{F}^h | x_{\zeta_r \eta_s}=1\}} (c^h + \lambda^h) \cdot x^h = \min_{\{\tilde{\mathbf{x}}^u \ | \ (\tilde{x}_\eta, \tilde{x}_\zeta)=(r,s)\}} \phi^u(\mathbf{x}^u), \tag{8.8}$$

for some $(\eta, \zeta) \in X_u$, $u \in I$, and the minimization of the right-hand side of (8.8) can be carried out for each $(r, s) \in R_\eta \times R_\zeta$ less inexpensively than that of the left-hand side.

Algorithm 11 summarizes the procedure to compute rotamer-pair lower bounds from the dual formulation. Computing rotamer lower bounds can be also similarly done by properly replacing step 7-8 with singleton rotamer assignment and min-sum values.

It should be noted that rotamer-pair lower bounds are computed only for pairs of positions η, and ζ such that $(\eta, \zeta) \in \mathcal{E}$. We did not have such a restriction for rotamer-pair lower bounds from TRMP in Section 3.3 because we could easily compute max-marginals for $(\eta, \zeta) \notin \mathcal{E}$ via enumeration when the trees were stars. This is not straightforward to be done on a general tree decomposition. However, when we can compute a rotamer or rotamer-pair lower bound based on the dual formulation, we know that it should be at least as tight as the rotamer or rotamer-pair lower bound from TRMP by Lemma 11.

Algorithm 11: Procedure for computing rotamer-pair lower bounds from the dual formulation.

```
  Data: $f(\mathbf{x})$, $(\{X_i \mid i \in I\}, T = (I, F))$ as in Algorithm 4.
  Result: $\phi^u(\mathbf{x}^u)$ for each $u \in I$, and $\lambda^{u\cap v}(\mathbf{x}^{u\cap v})$ for each $(u, v) \in F$.
1 begin
2     Solve problem (6.12) through the subgradient method to obtain $\hat{\lambda}$ at
      convergence of the subgradient method.
3     foreach $h$ do
4         Construct $f^h(\mathbf{x}^h; \hat{\lambda}^h)$ from the linearized cost function $(c^h + \hat{\lambda}^h) \cdot x^h$ and
          tree decomposition for $\mathcal{G}^h$.
5         Run the junction tree algorithm (Algorithm 8) on $f^h(\mathbf{x}^h; \hat{\lambda}^h)$.
6         Compute min-sums for each variable $i \in \mathcal{V}^h$ and pair of variables
          $(i, j) \in \mathcal{E}^h$ by enumerating node and separator messages from the
          junction tree algorithm.
7     foreach $(r, s) \in R_\eta \times R_\zeta$, $(\eta, \zeta) \in \mathcal{E}$ do
8         Compute the rotamer pair lower bound by substituting the min-sum
          value for the rotamer pair from subproblem $h$ if $(\eta, \zeta) \in \mathcal{E}^h$, and the
          minimum value of the subproblem $h$ if $(\eta, \zeta) \notin \mathcal{E}^h$ for the corresponding
          terms in the objective function of (8.7).
9 end
```

8.3 Computational experiments

In this section, we compare rotamer-pair lower bounds computed from TRMP and the dual method through computational experiments. We also incorporate the subgradient method and individual bounds computation under BroMAP, and evaluate the performance of the resulting branch-and-bound implementation.

We implemented Algorithm 8, 9, 10, and 11 in C++, compiled the code using GNU compiler version 3.3.5, and run the following experiments on a Linux workstation with a Dual-Core AMD Opteron Processor 2214, and 2 GBytes of memory.

8.3.1 TRMP vs. dual method

We compared the two methods of computing rotamer-pair lower bounds using a subproblem CASE46-NODE42 of Case 44 in Table 5.1. CASE46-NODE42 consists of 11 positions, 696 rotamers, and 208,816 rotamer pairs, and log of the total number of conformations is $\sum \log |R_i| = 18.7$. We varied the number of triplets used to derive

Table 8.1: Running times for computing rotamer pair lower bounds for all rotamer pairs in CASE46-NODE42. All times are shown in seconds. Either TRMP or the dual method was used. For the dual method, the number of triplets included in the formulation was varied. The total running time can be divided into time for running the TRMP or the dual method until convergence, and time for executing the post-process using Lemma 3 for TRMP or the junction tree algorithm for the dual method.

Method	# Triples	Total	Propagation	Post-process
TRMP	NA	10	8	2
Dual	0	126	125	1
Dual	2	255	254	1
Dual	4	678	677	1
Dual	6	1,038	1,036	2
Dual	8	1,180	1,177	3
Dual	10	1,468	1,465	3

a dual formulation. For TRMP, the stopping condition was made so that the maximum absolute change of pseudo-max-marginals becomes less than 10^{-7}. For the dual method, we used the stopping condition $\|\Delta\lambda\|_\infty < 10^{-4}$. It should be noted that, unlike TRMP, the dual method is guaranteed to generate lower bounds for even weaker stopping conditions. In the dual method, instead of using the junction tree method in the subgradient method for all iterations, we used the dynamic programming of Chapter 6 until convergence, and run one iteration of the junction tree method as a post-process. This is because the implementation of the dynamic programming was found to be more efficient than that of the junction tree algorithm during experiments.

Table 8.1 shows the running time for computing rotamer-pair lower bounds of all rotamer pairs in CASE46-NODE42, for each number of triplets and TRMP. The total running time of the bounding time can be divided into propagation time of TRMP or the dual method, and time for performing the post-process, that is, applying Lemma 3 for TRMP, or running the junction tree algorithm for the dual method. As noted previously in Chapter 6, the dual method has much longer propagation time than TRMP. However, the post-processing time of the dual method was found comparable to that of TRMP.

Figure 8-1 compares rotamer-pair lower bounds from the two methods. If we let $LB_{trmp}(i_r, j_s)$ the TRMP lower bound of rotamer pair (i_r, j_s), and $LB^N_{dual}(i_r, j_s)$

Table 8.2: Size information of subproblems of Case 46 used in the experiment with BroMAP using either TRMP or the dual method for bounding. The subproblems were generated from the first depth-first dive of BroMAP using the dual method. Column (1) # Positions: number of positions, (2) $\sum |R_i|$: total number of rotamers, (3) Pairs: total number of rotamer pairs, (4) $\log conf$: $\sum_{i=1}^{n} \log |R_i|$,

No.	Case	# Positions	$\sum \lvert R_i \rvert$	Pairs	$\log conf$
I CASE46-NODE50	11	462	95094	17.4	
II CASE46-NODE46	11	594	151024	18.1	
III CASE46-NODE42	11	696	208816	18.7	

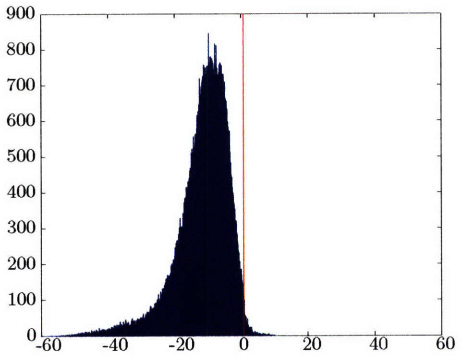
the dual lower bound from N triplets for the same rotamer pair, then each plot corresponds to the histogram of $LB^{N}_{dual}(i_r, j_s) - LB_{trmp}(i_r, j_s)$ for all rotamer pairs (i_r, j_s) in CASE46-NODE42 and N = 0, 2, 4, 6, 8, or 10. The line vertically crossing each plot shows $\min_{(i_r,j_s)} LB^{N}_{dual}(i_r, j_s) - \min_{(i_r,j_s)} LB_{trmp}(i_r, j_s)$, that is, the difference between the lower bounds from two methods for the entire subproblem CASE46-NODE42.

We can observe several trends from Figure 8-1. First, the histogram shifts rightward as we add triplets although we observe only slight difference between 8 triplets and 10 triplets. This obviously suggests rotamer-pair lower bounds are also strengthened as the dual formulation includes more triplets. Second, for all numbers of triplets plotted, there are many rotamers such that $LB^{N}_{dual}(i_r, j_s) - LB_{trmp}(i_r, j_s)$ is negative, that is, rotamers whose dual lower bounds are weaker than the TRMP lower bounds. Third, comparing the relative positioning of the large mass of the histogram and the vertical line suggests that the gains in rotamer-pair lower bounds from using the dual method instead of TRMP are not as large as the gain in the global lower bound.

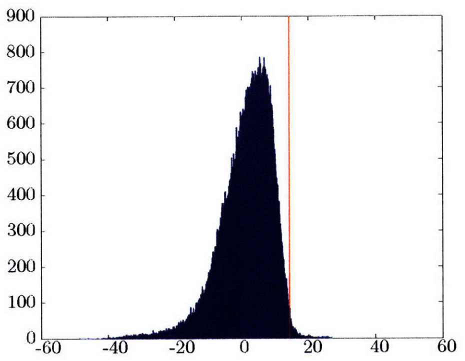
8.3.2 BroMAP with dual bounds

We applied the implementation of BroMAP using either TRMP or the dual method for bounding to subproblems of Case 46 in Table 5.1. Table 8.2 shows the size information of the subproblems used in the experiment.

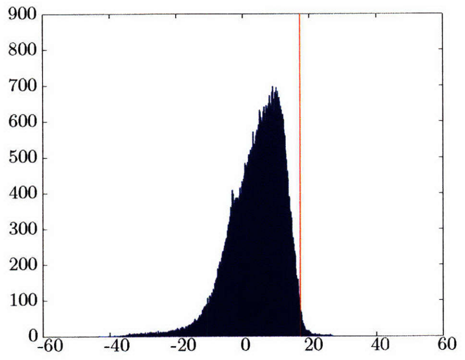
For each test case, we measured three running times, each with one of the followings:



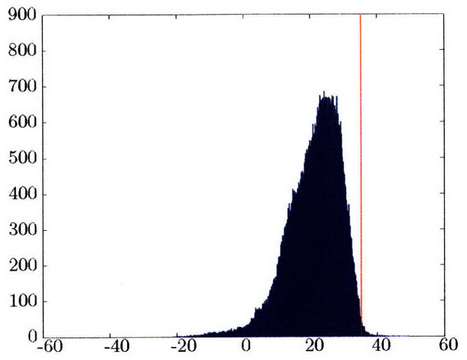
(a) No triplets.



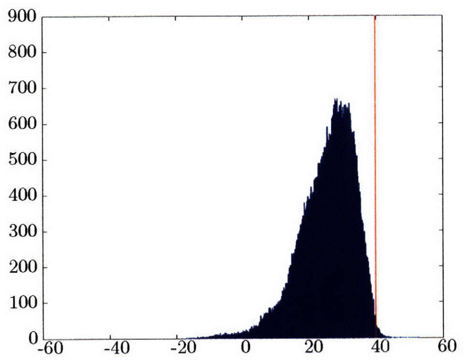
(b) Two triplets.



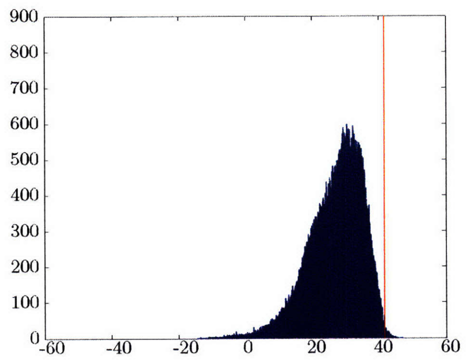
(c) Four triplets.



(d) 6 triplets.



(e) 8 triplets.



(f) 10 triplets.

Figure 8-1: Histogram of $LB_{dual}^{N}(i_r, j_s) - LB_{trmp}(i_r, j_s)$ for $N = 0$, 2, 4, 6, 8, and 10, where N is the number of triplets used for the dual formulation. The vertical line in each plot represents $\min_{(i_r,j_s)} LB_{dual}^{N}(i_r, j_s) - \min_{(i_r,j_s)} LB_{trmp}(i_r, j_s)$.

1. BroMAP using the dual method for bounding. Each node in the BroMAP tree is subjected to one run of *DEE-gp*, and upper/lower-bounding using the dual method. The number of triplets used in the dual formulation is equal to the boundability index of the node as defined in Chapter 5

2. BroMAP using TRMP with no rotamer contraction. Therefore, each node is subject to *DEE-gp* and upper/lower bounding by TRMP.

3. BroMAP using TRMP with rotamer contraction. The number of rotamer contractions performed in each node of the BroMAP tree is determined by 16 times the boundability index. Therefore, each node is possibly subject to multiple times of rotamer contraction, TRMP bounding, and *DEE-gp*.

The reason we use different number of triplets in the dual method is similar to the reason that the number of rotamer contractions is determined by the boundability index in Chapter 5; the running time of the dual method is larger than TRMP even for zero triplets. Thus, using a fixed number of triplets for all nodes of the BroMAP tree will significantly increase the running time over the BroMAP using TRMP. However, the lower bounds from the dual method will not be significantly stronger than TRMP lower bounds without the help of a good number of triplets, as was observed in Section 8.3.1. Therefore, we adaptively increase the number of triplets for the nodes that are likely to have high optimal values and be pruned easily.

The current implementation of BroMAP using the dual method does not include the feature of rotamer contraction. Therefore, by running BroMAP using TRMP without rotamer contraction, we can make a direct comparison of bounds from two methods in BroMAP by comparing the running times from method 1 and 2 above.

We fed an initial upper bound U^0 found from the initial branch-and-bound that generated the test cases for Case 46. For *DEE-gp*, we used the same combination of elimination conditions described in Chapter 5. A^* was allowed to run on nodes with log conformation smaller than 13 and, the number of pairs less than $100,000$.

Table 8.3 shows the results of solving the test cases in Table 8.2 using BroMAP with three different configurations. For all three test cases, the results suggest that

Table 8.3: Results of solving test cases of Table 8.2 with BroMAP using either TRMP or dual bounds. Columns (1) Case: test case, (2) Bro: BroMAP with combination of a bounding method and use/no use of rotamer contraction as described in the text, (3) T-Br: total number of branchings, (4) F-Br: number of branchings during the first depth-first dive, (5) Skew: skewness of the search tree defined as $\frac{(\text{number of low-subproblems split})}{(\text{total number of splits})-1}$, (5) F-Ub: $U_{F-BR} - U^0$, where U_{F-BR} is the upper bound obtained at the end of the first depth-first dive and U^0 is the upper bound fed as input, (6) average reduction of $\sum_i \log_{10} |R_i|$ during the first depth-first dive, i.e. (log*conf* - Leaf)/(F-Br), where log*conf* is defined in Table 8.2, (7) %RC: BroMAP time percentage used for rotamer contractions, (8) %DE: BroMAP time percentage used for *DEE-gp*, (9) %A^*: BroMAP time percentage used for A^*, (13) %PR: BroMAP time percentage used for propagation of TRMP or the dual method.

Case	Bro	T_{total}	T-Br	F-Br	Skew	F-Ub	Rdctn	%RC	%DE	%A^*	%PR
I	1	1.2E4	350	11	0.61	-1.8	0.37	NA	2	21	77
	2	3.6E3	304	11	0.59	-1.8	0.37	NA	13	68	19
	3	920	57	11	0.75	-1.8	0.37	7	18	47	28
II	1	4.6E4	1,439	13	0.57	17.1	0.39	NA	3	27	69
	2	1.5E4	1,240	10	0.58	3.7	0.45	NA	17	58	24
	3	2.5E3	55	10	0.83	3.7	0.45	7	15	57	21
III	1	8.5E4	3,174	16	0.57	2.0	1.17	NA	4	20	76
	2	2.6E4	2,114	16	0.61	2.0	1.17	NA	21	47	31
	3	3.3E3	131	16	0.81	2.0	1.17	10	23	30	37

BroMAP with TRMP bounds perform better than BroMAP with dual bounds. The main reason for increased running time when using dual bounds can be found from the increased bounding time as shown in the column '%PR' of Table 8.3. In addition, the fact that BroMAP using dual bounds needed more branchings than BroMAP using TRMP bounds (refer column 'T-Br') suggests that the segregation of rotamers using dual rotamer lower bounds was not as effective as that using TRMP rotamer lower bounds. Comparing the results from BroMAP with configuration 2 and 3, that is, cases using rotamer contraction and not, but always using TRMP bounds, it is observed that employing rotamer contraction improves the running time of BroMAP significantly.

8.4 Conclusions

In this chapter, we described a method based on the Hugin architecture junction tree algorithm that computes lower bounds for individual rotamers and rotamer pairs in the dual framework. The experimental comparison of rotamer pair lower bounds from the dual method and TRMP suggests that the improvement of the global lower bound attained by using the dual method instead of TRMP does not directly translate to the improvement of individual lower bounds. Instead, we found the rotamer pair lower bounds from the dual method are sometimes weaker than the bounds from TRMP even when the dual problem is derived from a number of triplets. We suspect the main difference comes from the fact that TRMP maintains and refines one pairwise pseudo-max-marginal value for each rotamer pair whereas the dual method computes a lower bound as an explicit convex combination of optimization results from each subproblem. This somewhat agrees with the previous observation of Chapter 6 that the global lower bound from the dual method for a fixed number of triplets weakens as we use more number of subproblems to cover the original graphical model and the triplets.

We also employed dual bounds in BroMAP and compared its performance with that of BroMAP employing TRMP bounds using small subproblems of a protein design case. Due to the long running time of the dual method and weak separation of rotamers by individual rotamer lower bounds from the dual method, BroMAP using dual bounds performed worse than BroMAP using TRMP bounds. It is our interest to further investigate ways of utilizing the dual method to improve the speed of BroMAP over the case of using TRMP bounds. Using alternatives to the subgradient methods, such as message-passing algorithms [Globerson and Jaakkola, 2007a] is expected to speed up the solution time of the dual problem. Adaptation of a fast message-passing dual method to additionally compute individual bounds can be an interesting future work. In addition, rotamer contraction can be very useful when used together with the dual method because we can incrementally use more expensive dual formulations as the problem size is reduced by rotamer contraction repeatedly.

Chapter 9

Conclusions

This thesis addresses the problem of exactly solving the global minimum energy conformation (GMEC) search problem exactly. We particularly focused on a specific instance, for which there were a larger number of rotamer choices at each design position and the interaction network between design position was dense. Such GMEC problems are often encountered in protein-protein interface design or protein core design. These problems impose the computational challenge that the de facto standard method called dead-end elimination (DEE) cannot handle very well with limited computational resources, even when augmented with the A^* algorithm.

The GMEC problem is equivalent to the MAP estimation problem studied for machine learning applications. In this work, we borrowed the techniques used for MAP estimation called tree-reweighted max-product (TRMP) algorithms to obtain systematic estimations of the GMEC energy. However, the large cardinality of conformational space often forbids direct application of these techniques to the GMEC problem. It is also often the case that estimation of the GMEC energy from simple uses of TRMP is not accurate enough to be useful. Therefore, we leveraged MAP estimation using a systematic search method called branch-and-bound, which tackles a small fraction of the entire problem at a time and seeks to exclude a subset of conformational subspace through bounds comparison. The final version of our method additionally incorporates DEE into the branch-and-bound framework, rotamers and rotamer-pairs elimination by TRMP bounds, and other techniques that produce lower bounds through simplification of the problem.

The resulting method, called BroMAP (branch-and-bound rotamer optimization through MAP estimation), showed promising results in our computational experiments using protein design cases. Our method extended the class boundary of protein design cases by solving large scale cases that could not be addressed using DEE/A^* on a typical workstation. It also outperformed DEE/A^* for challenging design cases.

In the second part of the thesis, we further explored several ways of improving the MAP estimation, that is, the lower bound of the GMEC energy, by using Lagrangian relaxation techniques. Typically, very good upper bounds are obtained relatively easily in the early stage of BroMAP at the end of the first depth-first dive. Therefore, a speed-up of BroMAP can be obtained if stronger lower bounds are available for similar cost of computation, which will result in more aggressive pruning of the search space without the loss of any accuracy.

We took an approach of using the subgradient method to directly solve the dual problem. A dual problem derived from sets of trees that cover the original graphical model will have the same optimal value as that of the linear program TRMP solves. Adding triplets to the formulation, the lower bounds from the dual problem are improved. We suggested a scheme for triplets selection based on strength of pairwise energies. Incremental addition of triplets to the formulation while the subgradient method continues was seen to reduce the running time of the subgradient method.

We further explored techniques for improving the lower bound computed by the subgradient method. In this approach, instead of increasing the number of triplets, the Lagrangian relaxation was tightened through addition of violated valid inequalities. Evaluation using small problems showed that the method can find the optimal value or improve the lower bound by repeatedly adding the inequalities to the dual problem and re-solving it. However, the computational cost for identifying relevant inequalities and re-solving the dual problem in the current scheme tends to be excessive for the degree of bounds improvement.

Just as using rotamer and rotamer-pair lower bounds from TRMP turned out to be a large improvement in BroMAP, the strengthened Lagrangian relaxation can be more usefully adopted in BroMAP by computing rotamer and rotamer-pair lower bounds. Such lower bounds can be readily computed from the result of Hugin propagation. Evaluation of these individual lower bounds and the dual method was made through integration with BroMAP. The preliminary results suggest that the individual bounds from the dual method are not necessarily stronger than those from TRMP. In addition, use of the dual bounds in BroMAP does not in general improve the running time of BroMAP. We still expect using different faster solution methods for the dual problem, such as message-passing algorithms, and combining rotamer contraction with the dual method can make BroMAP using dual bounds more competitive.

Future research in the direction of improving the running time of the dual method is called for. Compared to the compact algorithmic form of TRMP, the current subgradient method provides more flexibility in employing various optimization strategies, but the computation appears less efficient. The dual method described in this work also offers large room for improvement through, for example, more efficient coordination of dynamic programming applied to each subproblem, approximate computation of subgradients, more rigorous approximation of primal solutions, and optimization of separation routines.

The thesis lacks a sound mechanism of governing the overall running time of branch-and-bound in conjunction with the computation performed for each subproblem. For example, the dual method using addition of triplets or cuts can provide stronger bounds, but the time cost of doing them can be very expensive. A robust practical method will be made possible through systematically balancing the effect of obtaining accurate results from subproblems and the overall performance of branch-and-bound.

BroMAP was developed and evaluated with the specific class of protein design cases that need to be solved in mind. However, our method does not fully exploit any problem-specific information other than the discrete energy terms computed from the force fields model. Such energy terms are a complete description of the problem, but may be ineffective or too implicit in conveying certain types of constraints. For example, the computation can be simplified by incorporating geometric constraints derived from the physical context, such as the three-dimensional shape of the backbone structure and relative position of amino acids and rotamers.

The techniques described in this thesis are also applicable to other areas that require discrete MAP estimation on the graphical model. It is our interest to have a chance for evaluating the techniques on data from different applications. Finally, some techniques presented in this work may be less useful depending on the computing environment, or because of the expensive computations required to obtain incremental improvements. However, it is the author's belief that rigorous systematic approaches such as Lagrangian relaxation or convex approximation, in general, will become more useful as the computing environment evolves further.

Appendix A

Hardness results of the GMEC problem

The vertex weight can be incorporated in the edges incident to the vertex so that the GMEC problem only deals with edge weights. We modify the edge weights as follows:

$$w(i_r, j_s) = E(i_r j_s) + \frac{E(i_r) + E(j_s)}{k-1}, \tag{A.1}$$

where k is the number of residues in the protein. Obviously, the objective value is not affected by this modification. In this report, we assume $w(i_r, j_s)$ is nonnegative.

A.1 NP-hardness

Now we show hardness of the GMEC problem with nonnegative edge weights and no vertex weights.

Lemma 15. *The GMEC problem is strongly NP-hard.*

Proof. We show that the GMEC problem is NP-hard even when restricted to instances where every $E(i_r j_s)$ is restricted to 0 or 1 and $E(i_r)$ is 0. We describe a polynomial time reduction from $3SAT$. Given a CNF ϕ with k clauses, we construct an undirected graph G as follows: The vertices in G are organized into k groups of three vertices.

Each group corresponds to a clause in ϕ, and each vertex in the group to a literal in the associated clause. Edges of G connect every pair of vertices but those in a group. Each edge has weight 1 if one end point is inverse of the other, and 0 otherwise.

With this reduction, if ϕ is satisfiable, then there are at least one true literal in each clause. Taking one true literal from each clause constitutes a k-clique in G, and the edge weights in the clique are all 0 since no conflicting literals can be true at the same time. Therefore, the optimal sum of weights is 0.

As the reverse direction, if G has optimal sum of weights equal to 0, then all the edges in k-clique has weight 0. This suggests that there are no conflicting literals in the k-clique. By assigning true to literals in the clique, each clause has at least one true literal. Therefore, ϕ is satisfied. $\square$

We note that the GMEC problem is also strongly NP-hard when each part contains only two vertices. Using the same reduction from above on $MAX\text{-}2SAT$ with k clauses, we observe that $MAX\text{-}2SAT$ has optimal value m *iff* the GMEC problem has optimal value $\binom{k}{2} - \binom{m}{2}$. Furthermore, an optimal solution of the $MAX\text{-}2SAT$ is given by assigning true to vertices of the k-clique in the optimal solution of the GMEC problem. Since $MAX\text{-}2SAT$ is NP-hard, the GMEC problem whose part containing two vertices is also NP-hard.

A.2 Hardness of approximation

Using the same reduction from above, it is easily seen that the GMEC problem does not have a constant factor approximation algorithm unless $P = NP$. Suppose it has a c-approximation algorithm. Then, $3SAT$ can be determined in polynomial time to be satisfiable *iff* the c-approximation algorithm for the GMEC problem finds a solution value 0 for the constructed graph because for any $c > 0$, the worst approximation solution value for 0 is still 0.

A.3 Approximation for metric weights

We present a simple $\frac{k+1}{4}$-approximation algorithm for the metric instances of the GMEC problem. That is, when $w(x_r, z_t) > w(x_r, y_s) + w(y_s, z_t)$ is true for all vertices x_r, y_s, and z_t, $x \neq y \neq z$. We also assume that the number of parts, k, is a prime number.

In a k-clique where k is a prime number, we note that the clique can be decomposed into $\frac{k-1}{2}$ edge-disjoint tours of vertices in the clique. For example, when $k = 7$, suppose we label vertices as 1, 2, 3, 4, 5, 6, and 7. Then, we can decompose the 7-clique into 3 tours as follows:

Tour 1 only using edges $(l, l+1 \mod 7)$	$1-2-3-4-5-6-7-1$
Tour 2 only using edges $(l, (l+1) \mod 7)$	$1-3-5-7-2-4-6-1$
Tour 3 only using edges $(l, (l+2) \mod 7)$	$1-4-7-3-6-2-5-1$

In this way, after making $\frac{k-1}{2}$ separate tours, each only using edges $(i, (i+l \mod k)$, $l = 1, \ldots, \frac{k-1}{2}$, all edges in the clique are covered exactly once. This is because exactly k hops are necessary to come back to the starting point since k is a prime number, and each edge cannot belong to more than one tour by the construction of the tour, resulting in $\frac{k(k-1)}{2}$ distinct edges traversed, which is equal to the number of total edges in a k-clique.

Let W be the sum of edge weights in the k-clique, and $W_1, W_2, \ldots, W_{\frac{k-1}{2}}$ be sum of edge weights in tour $1, 2, \ldots, \frac{k-1}{2}$. Then, we note that

$$\min_i W_i \leq \frac{W}{\frac{k-1}{2}} \tag{A.2}$$

Based on this fact, we can think of a simple approximation algorithm. However, instead of tours of vertices, we consider tour of parts in the GMEC problem. That is, we are required to visit exactly one vertex in each part. The algorithm is as follows:

1. For each tour of parts T_i:

   (a) For each vertex v in the starting part of T_i

Run Dijkstra's shortest path algorithm to find the minimum weight tour $t_i(v)$ from v to v only using edges in T_i.

(b) Find $\min_v t_i(v)$ and let the associated tour $T_{i,min}$.

2. Find $\min_i T_{i,min}$ and let the associated tour T_{min}.

3. Add non-existing edges between every pair of vertices in T_{min} to construct a k-clique.

Figure A-1, Figure A-2 and Figure A-3 illustrate the algorithm. If we apply the suggested algorithm on the graph shown in Figure A-1, we iterate step 1 twice since there are two distinct tour of parts; $1-2-3-4-5$ and $1-3-5-2-4-1$. Figure A-2 gives a detailed illustration how a minimum weight tour is computed for each tour of parts. We run Dijkstra's algorithm on the first half of the graph (from part 1 to part 1′) for each vertex in part 1 as the starting point. We do the same for the rest of the graph. In the end, as shown in Figure A-3, we identify the minimum weight tour $(1_2-2_2-3_3-4_3-5_3-1_2)$, and add edges to find a feasible solution.

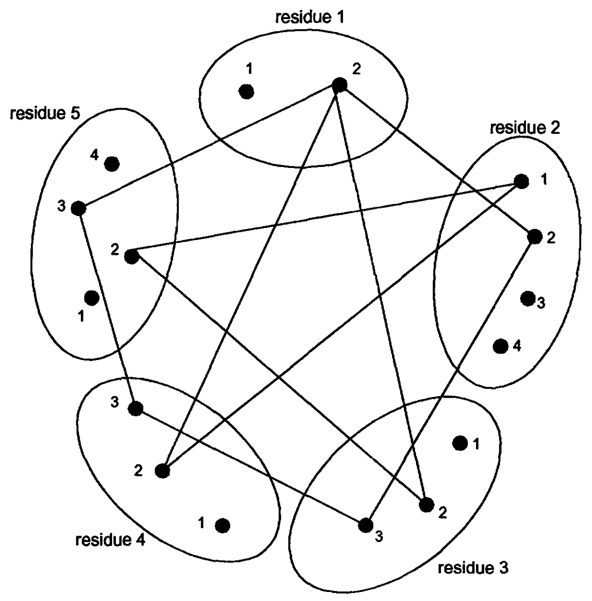


Figure A-1: Tour $1_2-2_2-3_3-4_3-5_3-1_2$ and tour $1_2-3_2-5_2-2_1-4_2-1_2$

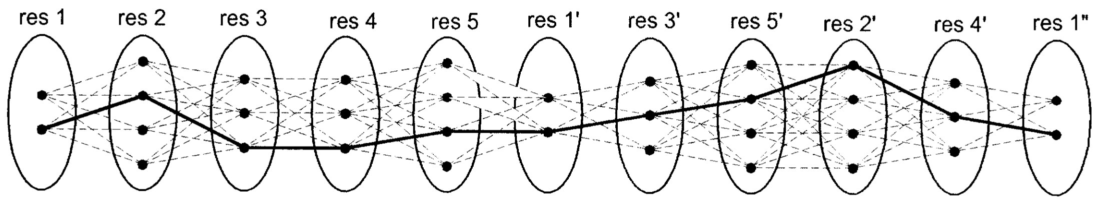


Figure A-2: An elongated graph to calculate minimum weight tours

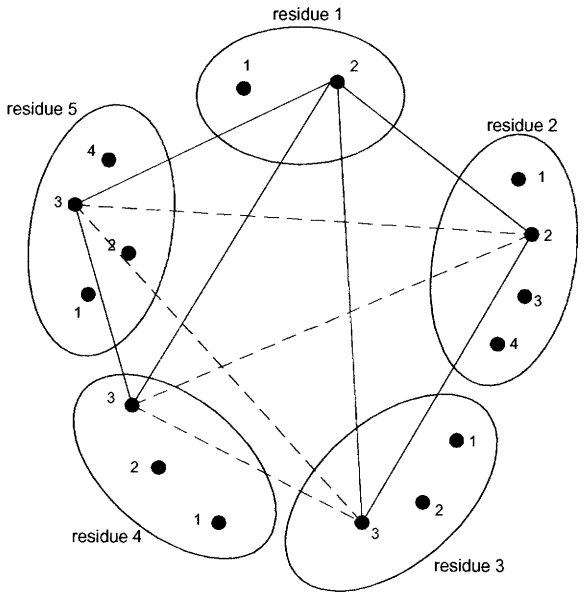


Figure A-3: A feasible solution to the GMEC problem

Now we show that the algorithm gives $\frac{k+1}{4}$-approximation. By the fact presented previously, we know, about $W(T_{min})$, the weight of T_{min}, that

$$W(T_{min}) \leq \frac{2OPT}{k-1} \tag{A.3}$$

We can arbitrarily relabel the vertices in the tour. Suppose we labeled vertices so that T_{min} is $v_1 - v_2 - v_3 - \ldots - v_k - v_1$. By jumping over one adjacent vertex in T_{min}, we add edges of a tour $v_1 - v_3 - v_5 - \ldots - v_{k-1} - v_1$. By the metric assumption, we know that

$$\begin{aligned}
\sum_{i=1}^{k} w(v_i, v_{(i+2 \mod k)}) &= w(v_1, v_3) + w(v_3, v_5) + \ldots w(v_{k-1}, v_1) && \text{(A.4)}\\
&\leq (w(v_1, v_2) + w(v_2, v_3)) + (w(v_3, v_4) + w(v_4, v_5)) + \ldots \\
&\quad +(w(v_{k-1}, v_k) + w(v_k, v_1)) && \text{(A.5)}\\
&= 2\sum_{i=1}^{k} w(v_i, v_{(i+1 \mod k)}) && \text{(A.6)}\\
&= 2W(T_{min}) && \text{(A.7)}
\end{aligned}$$

In the same fashion, we can show, in general,

$$\begin{aligned}
\sum_{i=1}^{k} w(v_i, v_{(i+l \mod k)}) &\leq l\sum_{i=1}^{k} w(v_i, v_{(i+1 \mod k)}) && \text{(A.8)}\\
&= lW(T_{min}) && \text{(A.9)}
\end{aligned}$$

Therefore, we know, about $W(C)$, the weight of the constructed clique C by the algorithm, that

$$\begin{aligned}
W(C) &\leq \sum_{l=1}^{\frac{k-1}{2}} \sum_{i=1}^{k} w(v_i, v_{(i+1 \mod k)}) && \text{(A.10)}\\
&\leq \sum_{l=1}^{\frac{k-1}{2}} lW(T_{min}) && \text{(A.11)}\\
&\leq \frac{(k-1)(k+1)}{8} W(T_{min}) && \text{(A.12)}
\end{aligned}$$

Finally, from (A.3), we have $W(C) \leq \frac{k+1}{4} OPT$.

The algorithm runs in polynomial time $kO(n)O(\text{Dijkstra})) = O(kn^3)$.

Appendix B

Algorithm descriptions

B.1 TRMP

Algorithm 12 describes "edge-based reparameterization updates" [Wainwright et al., 2005] for a set of general trees $\mathcal{T}$. In line 2, 3, 5, and 6, κ_i^n and κ_{ij}^n are constants that can be arbitrarily set as long as they are positive. $\Gamma(i)$ is the set of vertices neighboring i in $\mathcal{G}$ for $i \in \mathcal{V}(\mathcal{G})$.

Algorithm 12: TRMP (edge-based reparameterization updates [Wainwright et al., 2005, Kolmogorov, 2006])

```
  **Data**: $\epsilon > 0$, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\{\rho\}$, $\{e\}$
  **Result**: tree-consistent pseudo-max-marginals $\{\nu\}$
1 begin
2     $\nu_i^0(x_i) \leftarrow \kappa_i^0 \exp\left(-\frac{e_i(x_i)}{\rho_i}\right), \quad i \in \mathcal{V}$
3     $\nu_{ij}^0(x_i, x_j) \leftarrow \kappa_{ij}^0 \exp\left(-\frac{e_{ij}(x_i,x_j)}{\rho_{ij}} - \frac{e_i(x_i)}{\rho_i} - \frac{e_j(x_j)}{\rho_j}\right), \quad (i,j) \in \mathcal{E}$
4     repeat update pseudo-max-marginals
5         $\nu_i^{n+1}(x_i) \leftarrow \kappa_i^{n+1} \nu_i^n(x_i) \prod_{j \in \Gamma(i)} \left(\frac{\max_{x_j \in R_j} \nu_{ij}^n(x_i,x_j)}{\nu_i^n(x_i)}\right)^{\frac{\rho_{ij}}{\rho_i}}, \quad i \in \mathcal{V}$
6         $\nu_{ij}^{n+1}(x_i, x_j) \leftarrow \kappa_{ij}^{n+1} \frac{\nu_{ij}^n(x_i,x_j)\nu_i^{n+1}(x_i)\nu_j^{n+1}(x_j)}{\max_{x_j' \in R_j} \nu_{ij}^n(x_i,x_j') \max_{x_i' \in R_i} \nu_{ij}^n(x_i',x_j)}, \quad (i,j) \in \mathcal{E}.$
7     until $|\nu^{n+1} - \nu^n| < \epsilon$, update $n \leftarrow n + 1$
8 end
```

B.2 Subprocedures of Algorithm 14

Algorithm 11 and Algorithm 13 describe the subprocedures of *DEE-gp*.

```
Algorithm 13: eliminate-singles
  Data: $\{R_i\}, \{\hat{e}\}, \hat{D}, \hat{P}, \{e\}, \tilde{P}, \zeta, \eta$
  Result: $\{R'_i\}, \{\hat{e}'\}, \hat{P}', \{e'\}, \tilde{P}'$
1 begin
2     foreach $i_r \in R_i,\ i \in \mathcal{V}$ do additional singles-flagging using pair-flags
3         foreach $j \in \mathcal{V},\ j \neq i$ do
4             if $R_j = \{j_s | (i_r, j_s) \in \hat{P} \cup \tilde{P}\}$ then $\hat{D} \leftarrow \hat{D} \cup \{i_r\}$
5     foreach $i \in \mathcal{V}$ do define $R'_i$
6         $R'_i \leftarrow R_i \backslash \{i_r | i_r \in \hat{D}\}$
7     if $R'_i = \phi$ then
8         the problem is infeasible, so quit with optimal value $+\infty$
9     else leave the terms of $\{\hat{e}'\}$ and $\{e'\}$ only for $\{R'_i\}$, and
10        $\hat{P}' \leftarrow \hat{P} \backslash \{(i_r, j_s) | i_r \in \hat{D}$ or $j_s \in \hat{D}\}$, $\tilde{P}' \leftarrow \tilde{P} \backslash \{(i_r, j_s) | i_r \in \hat{D}$ or $j_s \in \hat{D}\}$
11 end
```

B.3 ILP formulation

The ILP formulation for the GMEC problem referred in this paper is as follows:

$$\min_{\{x_{i_r}\},\{x_{i_r j_s}\}} \left[\sum_{i\in\mathcal{V}} \sum_{r\in R_i} E(i_r) x_{i_r} + \sum_{(i,j)\in\mathcal{E}} \sum_{(r,s)\in R_i\times R_j} E(i_r j_s) x_{i_r j_s} \right] \tag{B.1}$$

$$\sum_{r\in R_i} x_{i_r} = 1, \quad \forall i \in \mathcal{V} \tag{B.2}$$

$$\sum_{s\in R_j} x_{i_r j_s} = x_{i_r}, \quad \forall (i,j) \in \mathcal{E},\ \forall r \in R_i, \tag{B.3}$$

$$x_{i_r} \in \{0,1\}, \quad \forall i \in \mathcal{V},\ r \in R_i, \tag{B.4}$$

$$x_{i_r j_s} \in \{0,1\}, \quad \forall (i,j) \in \mathcal{E},\ (r,s) \in R_i \times R_j. \tag{B.5}$$

An LP relaxation can be obtained by simply dropping the constraint (B.4)–(B.6). The resulting LP is equivalent to the tree-relaxed LP [Wainwright et al., 2005]. Note

Algorithm 14: unification

```
    Data: $\mathcal{G}, \{R_i\}, \{\hat{e}\}, \hat{P}, \{e\}, \tilde{P}$
    Result: $\mathcal{G}', \{R_i'\}, \{\hat{e}'\}, \hat{P}', \{e'\}, \tilde{P}'$
1   begin
        /* define $\mathcal{G}' \leftarrow (\mathcal{V}', \mathcal{E}')$                                */
2       $\mathcal{V}' \leftarrow (\mathcal{V} \backslash \{\zeta, \eta\}) \cup \{[\zeta : \eta]\}$
3       $\mathcal{E}' \leftarrow \{(i,j) \in \mathcal{E} | i,j \neq \zeta, \eta\} \cup \{([\zeta : \eta], j) | (\zeta, j) \in \mathcal{E}$ or $(\eta, j) \in \mathcal{E}\}$
        /* define $\{R_i'\}$, $i \in \mathcal{V}'$ the same as $\{R_i\}$ except             */
4       $R'_{[\zeta:\eta]} \leftarrow \{\ [\zeta_r : \eta_s] \mid (\zeta_r, \eta_s) \in R_\zeta \times R_\eta, (\zeta_r, \eta_s) \notin \hat{P} \cup \tilde{P}\}$
        /* define $\hat{P}'$ and $\tilde{P}'$                                            */
5       $\hat{P}' \leftarrow \{(i_r, j_s) | (i_r, j_s) \in \hat{P}, (i,j) \in \mathcal{E}', i, j \neq [\zeta : \eta]\}$
6       $\hat{P}' \leftarrow \hat{P}' \cup \{([\zeta : \eta]_{[r:s]}, j_t) | (\zeta_r, j_t) \in \hat{P}$ or $(\eta_s, j_t) \in \hat{P}\}$
7       $\tilde{P}' \leftarrow \{(i_r, j_s) | (i_r, j_s) \in \tilde{P}, (i,j) \in \mathcal{E}', i, j \neq [\zeta : \eta]\}$
8       $\tilde{P}' \leftarrow \tilde{P}' \cup \{([\zeta : \eta]_{[r:s]}, j_t) | (\zeta_r, j_t) \in \tilde{P}$ or $(\eta_s, j_t) \in \tilde{P}\}$
        /* define $\{\hat{e}'\}$ the same as $\{\hat{e}\}$ except                        */
9       $\hat{e}'_{[\zeta:\eta]}([r : s]) \leftarrow \hat{e}_\zeta(r) + \hat{e}_\eta(s) + \hat{e}_{\zeta\eta}(r,s)$ for $[r : s] \in R'_{[\zeta:\eta]}$
10      $\hat{e}'_{[\zeta:\eta]j}([r : s], t) \leftarrow \hat{e}_{\zeta j}(r,t) + \hat{e}_{\eta j}(s,t)$ for $[r : s] \in R'_{[\zeta:\eta]}$, $t \in R_j$, $j \in \mathcal{V}' \backslash [\zeta : \eta]$
        /* define $\{e'\}$ the same as $\{e\}$ except                                */
11      $e'_{[\zeta:\eta]}([r : s]) \leftarrow e_\zeta(r) + e_\eta(s) + e_{\zeta\eta}(r,s)$ for $[r : s] \in R'_{[\zeta:\eta]}$
12      $e'_{[\zeta:\eta]j}([r : s], t) \leftarrow e_{\zeta j}(r,t) + e_{\eta j}(s,t)$ for $[r : s] \in R'_{[\zeta:\eta]}$, $t \in R_j$, $j \in \mathcal{V}' \backslash [\zeta : \eta]$
13  end
```

that the constraint (B.6) can be replaced with

$$x_{i_r j_s} \geq 0, \quad \forall (i,j) \in \mathcal{E}, \ (r,s) \in R_i \times R_j. \tag{B.6}$$

without affecting the optimal value.

Appendix C

Proofs

C.1 Proof of Lemma 1

From (2.30), we have

$$\max_{\mathbf{x}} p(\mathbf{x}) = \max_{\mathbf{x}} \nu_c \prod_{S \in \mathcal{S}} \{p^S(\mathbf{x};\nu)\}^{\rho(S)} \leq \nu_c \prod_{S \in \mathcal{S}} \left\{\max_{\mathbf{x}} p^S(\mathbf{x};\nu)\right\}^{\rho(S)} \tag{C.1}$$

Since ν is tree-consistent with every $S \in \mathcal{S}$, we can easily find a MAP assignment $\mathbf{x}^S$ such that $\mathbf{x}^S \in \arg\max_{\mathbf{x}} p^S(\mathbf{x};\nu)$. For how to obtain such an assignment, please refer Wainwright et al. [2004] Then, due to the assumption that ν is tree-consistent with S and is in a normal form, we have the following properties:

$$\nu_i(x_i^S) = 1, \quad \text{for all } i \in \mathcal{V}(S), \tag{C.2}$$

$$\nu_{ij}(x_i^S, x_j^S) = 1, \quad \text{for all } (i,j) \in \mathcal{E}(S). \tag{C.3}$$

Therefore,

$$\max_{\mathbf{x}} p^S(\mathbf{x};\nu) = p^S(\mathbf{x}^S;\nu) = \prod_{i \in \mathcal{V}(S)} \nu_i(x_i^S) \prod_{(i,j) \in \mathcal{E}(S)} \frac{\nu_{ij}(x_i^S, x_j^S)}{\nu_i(x_i^S)\nu_j(x_j^S)} = 1. \tag{C.4}$$

Since (C.4) is true for every $S \in \mathcal{S}$ and $\sum_{S \in \mathcal{S}} \rho(S) = 1$, we obtain $\max_{\mathbf{x}} p(\mathbf{x}) \leq \nu_c$ from (C.1).

C.2 Proof of Lemma 2

From (2.30), we have

$$\begin{aligned}\max_{\{\mathbf{x}|x_\zeta=r\}} p(\mathbf{x}) &= \max_{\{\mathbf{x}|x_\zeta=r\}} \nu_c \prod_{S\in\mathcal{S}} \{p^S(\mathbf{x};\nu)\}^{\rho(S)} \le \nu_c \prod_{S\in\mathcal{S}} \left\{\max_{\{\mathbf{x}|x_\zeta=r\}} p^S(\mathbf{x};\nu)\right\}^{\rho(S)} \\ &= \nu_c \prod_{S\in\mathcal{S}:\zeta\in\mathcal{V}(S)} \left\{\max_{\{\mathbf{x}|x_\zeta=r\}} p^S(\mathbf{x};\nu)\right\}^{\rho(S)} \prod_{S\in\mathcal{S}:\zeta\notin\mathcal{V}(S)} \left\{\max_{\{\mathbf{x}|x_\zeta=r\}} p^S(\mathbf{x};\nu)\right\}^{\rho(S)} \quad \text{(C.5)}\end{aligned}$$

By the definition of max-marginals and the assumption that ν is tree-consistent, for $S \in \mathcal{S}$ such that $\zeta \in \mathcal{V}(S)$, we have

$$\nu_\zeta(r) = \kappa_\zeta \max_{\{\mathbf{x}|x_\zeta=r\}} p^S(\mathbf{x};\nu), \tag{C.6}$$

for some constant κ_ζ. We know there exists $r^* \in R_\zeta$ such that

$$\nu_\zeta(r^*) = \max_{x_\zeta\in R_\zeta} \nu_\zeta(x_\zeta) = \kappa_\zeta \max_{\{\mathbf{x}|x_\zeta=r^*\}} p^S(\mathbf{x};\nu). \tag{C.7}$$

Then, since ν is in a normal form, $\nu_\zeta(r^*) = 1$. We know from (C.4) in the proof of Lemma 1, that $\max_{\{\mathbf{x}|x_\zeta=r^*\}} p^S(\mathbf{x};\nu) = \max_{\mathbf{x}} p^S(\mathbf{x};\nu) = 1$. Therefore, $\kappa_\zeta = 1$, and

$$\max_{\{\mathbf{x}|x_\zeta=r\}} p^S(\mathbf{x};\nu) = \nu_\zeta(r). \tag{C.8}$$

On the other hand, for all $S \in \mathcal{S}$ such that $\zeta \notin \mathcal{V}(S)$, we know $\max_{\{\mathbf{x}|x_\zeta=r\}} p^S(\mathbf{x};\nu) = \max_{\mathbf{x}} p^S(\mathbf{x};\nu) = 1$. Plugging the obtained values of $\max_{\{\mathbf{x}|x_\zeta=r\}} p^S(\mathbf{x};\nu)$ and $\sum_{S\in\mathcal{S}:\zeta\in\mathcal{V}(S)} \rho(S) = \rho_\zeta$ into (C.5), we obtain $\max_{\{\mathbf{x}|x_\zeta=r\}} p(\mathbf{x}) \le \nu_c \nu_\zeta(r)^{\rho_\zeta}$.

C.3 Proof of Lemma 3

1. If $\zeta, \eta \neq \mathcal{V}(S)$, we know $\max_{\{\mathbf{x}|x_\zeta=r,x_\eta=s\}} p^S(\mathbf{x};\nu) = \max_{\mathbf{x}} p^S(\mathbf{x};\nu)$. Then, since ν is tree-consistent and in a normal form, $\max_{\{\mathbf{x}|x_\zeta=r,x_\eta=s\}} p^S(\mathbf{x};\nu) = 1$ from (C.4).

2. If $\zeta \in \mathcal{V}(S)$, and $\eta \notin \mathcal{V}(S)$, we know $\max_{\{\mathbf{x}|x_\zeta=r,x_\eta=s\}} p^S(\mathbf{x};\nu) = \max_{\{\mathbf{x}|x_\zeta=r\}} p^S(\mathbf{x};\nu)$. Then, from (C.8), $\max_{\{\mathbf{x}|x_\zeta=r,x_\eta=s\}} p^S(\mathbf{x};\nu) = \nu_\zeta(r)$

3. If $(\zeta,\eta) \in \mathcal{E}(S)$, by the definition of max-marginals and the assumption that ν is tree-consistent with every S, we have $\nu_{\zeta\eta}(r,s) = \kappa_{\zeta\eta} \max_{\{\mathbf{x}|(x_\zeta,x_\eta)=(r,s)\}} p^S(\mathbf{x};\nu)$, for some constant $\kappa_{\zeta\eta}$. We also know there exists $(r^*,s^*) \in R_\zeta \times R_\eta$ such that $\nu_{\zeta\eta}(r^*,s^*) = \max_{(x_\zeta,x_\eta)\in R_\zeta\times R_\eta} \nu_{\zeta\eta}(x_\zeta,x_\eta) = \kappa_{\zeta\eta} \max_{\{\mathbf{x}|(x_\zeta,x_\eta=(r^*,s^*))\}} p^S(\mathbf{x};\nu)$. Then, since ν is in a normal form, i.e. $\nu_{\zeta\eta}(r^*,s^*) = 1$ and we have

$$\max_{\{\mathbf{x}|(x_\zeta,x_\eta)=(r^*,s^*)\}} p^S(\mathbf{x};\nu) = \max_{\mathbf{x}} p^S(\mathbf{x};\nu) = 1 \tag{C.9}$$

from (C.4), we obtain $\kappa_{\zeta\eta} = 1$. Therefore, $\max_{\{\mathbf{x}|(x_\zeta,x_\eta=(r,s))\}} p^S(\mathbf{x};\nu) = \nu_{\zeta\eta}(r,s)$.

4. If $\zeta,\eta \in \mathcal{V}(S)$ and $(\zeta,\eta) \notin \mathcal{E}(S)$, let $\xi = \gamma(S)$. Then,

$$\max_{\{\mathbf{x}|(x_\zeta,x_\eta)=(r,s)\}} p^S(\mathbf{x};\nu) = \max_{\{\mathbf{x}|(x_\zeta,x_\eta)=(r,s)\}} \nu_\xi(x_\xi)\nu_\zeta(r)\nu_\eta(s)\frac{\nu_{\xi\zeta}(x_\xi,r)}{\nu_\xi(x_\xi)\nu_\zeta(r)} \tag{C.10}$$

$$\times\frac{\nu_{\xi\eta}(x_\xi,s)}{\nu_\xi(x_\xi)\nu_\eta(s)} \prod_{j\in\mathcal{V}(S)\backslash\{\zeta,\eta,\xi\}} \nu_j(x_j)\frac{\nu_{\xi j}(x_\xi,x_j)}{\nu_\xi(x_\xi)\nu_j(x_j)} \tag{C.11}$$

$$= \max_{\{\mathbf{x}|(x_\zeta,x_\eta)=(r,s)\}} \frac{\nu_{\xi\zeta}(x_\xi,r)\nu_{\xi\eta}(x_\xi,s)}{\nu_\xi(x_\xi)} \prod_{j\in\mathcal{V}(S)\backslash\{\zeta,\eta,\xi\}} \frac{\nu_{\xi j}(x_\xi,x_j)}{\nu_\xi(x_\xi)} \tag{C.12}$$

$$= \max_{x_\xi}\left\{\frac{\nu_{\xi\zeta}(x_\xi,r)\nu_{\xi\eta}(x_\xi,s)}{\nu_\xi(x_\xi)} \prod_{j\in\mathcal{V}(S)\backslash\{\zeta,\eta,\xi\}} \max_{x_j}\frac{\nu_{\xi j}(x_\xi,x_j)}{\nu_\xi(x_\xi)}\right\}. \tag{C.13}$$

From tree-consistency of ν, we have $\nu_\xi(x_\xi) = \kappa_{\xi j}\max_{x_j}\nu_{\xi j}(x_\xi,x_j)$. Since ν is in a normal form, for some $\mathbf{x}^S \in \arg\max_{\mathbf{x}} p^S(\mathbf{x};\nu)$, we have $\nu_\xi(x_\xi^S) = 1$, and $\max_{x_j}\nu_{\xi j}(x_\xi^S,x_j) = \nu_{\xi j}(x_\xi^S,x_j^S) = 1$. Therefore, $\kappa_{\xi j} = 1$, and $\nu_\xi(x_\xi) = \max_{x_j}\nu_{\xi j}(x_\xi,x_j)$. So the maximization over x_j in the parentheses of (C.13) is equal to 1 for all $j \in \mathcal{V}(S)\backslash\{\zeta,\eta,\xi\}$. Therefore,

$$\max_{\{\mathbf{x}|(x_\zeta,x_\eta)=(r,s)\}} p^S(\mathbf{x};\nu) = \max_{x_\xi}\frac{\nu_{\xi\zeta}(x_\xi,r)\nu_{\xi\eta}(x_\xi,s)}{\nu_\xi(x_\xi)}.$$

C.4 Proof of Lemma 4

1. Let $\mathbf{y} \in \mathcal{L}(\mathcal{X}, \tilde{P}')$. Then, by (4.1) and $\tilde{P}' \supset \tilde{P}$, we have $\tilde{g}(\mathbf{y}, \tilde{P}) = 0$, therefore $\mathbf{y} \in \mathcal{L}(\mathcal{X}, \tilde{P})$. Since this is true for all $\mathbf{y} \in \mathcal{L}(\mathcal{X}, \tilde{P}')$, we have $\mathcal{L}(\mathcal{X}, \tilde{P}') \subset \mathcal{L}(\mathcal{X}, \tilde{P})$. Therefore, either $\mathcal{L}(\mathcal{X}, \tilde{P}')$ is empty, or we have $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P}')} e(\mathbf{x}) \geq \min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x})$.

2. Let $\hat{\mathbf{x}} \in \arg\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x})$. Then, from $e(\hat{\mathbf{x}}) \leq U$, we know $\min_{\{\mathbf{x}|(x_i,x_j)=(\hat{x}_i,\hat{x}_j)\}} e(\mathbf{x}) \leq U$ for all $i, j \in \mathcal{V}$, $i \neq j$. So we have $(\hat{x}_i, \hat{x}_j) \notin P(\{e\}, U)$ for all $i, j \in \mathcal{V}$, $i \neq j$. Therefore, we know $\tilde{g}(\hat{\mathbf{x}}, \tilde{P}) = \tilde{g}(\hat{\mathbf{x}}, P(\{e\}, U)) = 0$, which combined with $\tilde{P}' \backslash \tilde{P} \subset P(\{e\}, U)$ gives $\tilde{g}(\hat{\mathbf{x}}, \tilde{P}') = 0$. Finally, from $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x}) \leq \min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P}')} e(\mathbf{x})$, we obtain $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P}')} = e(\hat{\mathbf{x}})$.

3. From $\mathcal{L}(\mathcal{X}, \tilde{P}') = \phi \subset \mathcal{L}(\mathcal{X}, \tilde{P})$, $\mathcal{L}(\mathcal{X}, \tilde{P})$ can be also empty. Otherwise, let $\hat{\mathbf{x}} \in \arg\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x})$. By contradiction: if we assume $e(\hat{\mathbf{x}}) \leq U$, we know $\tilde{g}(\hat{\mathbf{x}}, \tilde{P}) = \tilde{g}(\hat{\mathbf{x}}, P(\{e\}, U)) = 0$. Then, from $\tilde{P}' \backslash \tilde{P} \subset P(\{e\}, U)$, we also have $\tilde{g}(\hat{\mathbf{x}}, \tilde{P}') = 0$, which contradicts $\hat{\mathbf{x}} \in \mathcal{L}(\mathcal{X}, \tilde{P}') = \phi$.

C.5 Proof of Corollary 1

($\Leftarrow$) If $\mathcal{L}(\mathcal{X}, \tilde{P}) = \phi$, then $\mathcal{L}(\mathcal{X}, \tilde{P}') = \phi$. If $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x}) > U$, then Lemma 4.1 applies.

($\Rightarrow$) If $\mathcal{L}(\mathcal{X}, \tilde{P}') = \phi$, then Lemma 4.3 applies. If $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P}')} e(\mathbf{x}) > U$, then $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x})$ also exists. From $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P}')} e(\mathbf{x}) \geq \min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x})$, we divide the cases as follows:

1. if $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P}')} e(\mathbf{x}) = \min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x})$, then $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x}) > U$.

2. if $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P}')} e(\mathbf{x}) > \min_{\mathbf{x}} e(\mathbf{x})$, then $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x}) > U$ from Lemma 4.2.

C.6 Proof of Lemma 5

Let $\mathcal{Y}$ be the conformation space $\mathcal{X}$ or $\mathcal{X}^{rc}$ excluding position ζ, i.e.

$$\mathcal{Y} = R_1 \times R_2 \times \ldots \times R_{\zeta-1} \times R_{\zeta+1} \times \ldots \times R_n. \tag{C.14}$$

To simplify the notation, we define $\tilde{e}(\mathbf{x})$ as $\tilde{e}_\zeta(x_\zeta) = 0$ and $\tilde{e}_{\zeta j}(x_\zeta, x_j) = e_{\zeta j}(r, s) + \frac{e_\zeta(r)}{|\Gamma(\zeta)|}$ for $j \in \Gamma(\zeta)$, whereas all the other terms of $\tilde{e}(\mathbf{x})$ are same as $e(\mathbf{x})$. Then, we have $\tilde{e}(\mathbf{x}) = e(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$. We also use short notations $e^{rc}(\mathbf{x}) = e^{rc}(\mathbf{x}, \tilde{P})$.

1. We divide the proof into small parts as follows:

   (a) We show $\{\mathbf{y} \in \mathcal{Y} | \tilde{g}(\{\mathbf{y}, x_\zeta = r\}, \tilde{P}) = 0\} \subset \{\mathbf{y} \in \mathcal{Y} | \tilde{g}(\{\mathbf{y}, x_\zeta = c_k\}, \tilde{P}^{rc}) = 0\}$ is true for all $r \in C_k$, for each $k = 1, \ldots, l$.

   Let $L_r = \{\mathbf{y} \in \mathcal{Y} | \tilde{g}(\{\mathbf{y}, x_\zeta = r\}, \tilde{P}) = 0\}$ and $M_k = \{\mathbf{y} \in \mathcal{Y} | \tilde{g}(\{\mathbf{y}, x_\zeta = c_k\}, \tilde{P}^{rc}) = 0\}$. Let $\hat{\mathbf{x}} = \{\hat{\mathbf{y}}, \hat{x}_\zeta = r\}$, where $\hat{\mathbf{y}} \in L_r$. Then, we know $\tilde{g}_{ij}(\hat{x}_i, \hat{x}_j, \tilde{P}) = 0$ for all $(i, j) \in \mathcal{E}$. Therefore, we know $\tilde{g}_{ij}(\hat{x}_i, \hat{x}_j, \tilde{P}^{rc}) = \tilde{g}_{ij}(\hat{x}_i, \hat{x}_j, \tilde{P}) = 0$, for $i, j \neq \zeta$ by definition. For $j \in \Gamma(\zeta)$, we have $\tilde{g}_{\zeta j}(c_k, \hat{x}_j, \tilde{P}^{rc}) = \prod_{\tilde{r} \in C_k} \tilde{g}_{\zeta j}(\tilde{r}, \hat{x}_j, \tilde{P}) = 0$ because $r \in C_k$ and $\tilde{g}_{\zeta j}(r, \hat{x}_j, \tilde{P}) = 0$. So we obtain $\tilde{g}(\{\hat{\mathbf{y}}, x_\zeta = c_k\}, \tilde{P}^{rc}) = 0$ and $\hat{\mathbf{y}} \in M_k$. Therefore, $L_r \subset M_k$.

   (b) We show, if $\mathcal{L}(\mathcal{X}, \tilde{P})$ is not empty, then $\mathcal{L}(\mathcal{X}^{rc}, \tilde{P}^{rc})$ is not empty.

   We know

$$\mathcal{L}(\mathcal{X}, \tilde{P}) = \bigcup_{k=1,\ldots,l} \bigcup_{r \in C_k} \{\{\mathbf{y}, x_\zeta = r\} | \mathbf{y} \in L_r\}, \tag{C.15}$$

$$\mathcal{L}(\mathcal{X}^{rc}, \tilde{P}^{rc}) = \bigcup_{k=1,\ldots,l} \{\{\mathbf{y}, x_\zeta = c_k\} | \mathbf{y} \in M_k\}, \tag{C.16}$$

   where L_r and M_k are as defined in (1a). Since $\mathcal{L}(\mathcal{X}, \tilde{P})$ is not empty, L_t is not empty for some $t \in A_q$, for some q, $1 \leq q \leq l$. From (1a), we know $L_t \subset M_q$. Therefore, $\{\{\mathbf{y}, x_\zeta = \alpha_q\} | \mathbf{y} \in M_q\}$ is not empty. So $\mathcal{L}(\mathcal{X}^{rc}, \tilde{P}^{rc})$ is not empty either.

2. We divide the proof into small parts as follows:

(a) Following inequality is true for arbitrary finite functions $f(\mathbf{x})$ and $g(\mathbf{x})$, both defined over some non-empty finite set A:

$$\min_{\mathbf{x}\in A} f(\mathbf{x}) - \min_{\mathbf{x}\in A} g(\mathbf{x}) \geq \min_{\mathbf{x}\in A}\{f(\mathbf{x}) - g(\mathbf{x})\}. \tag{C.17}$$

We show (C.17) as follows: let $\tilde{\mathbf{x}} \in \arg\min_{\mathbf{x}\in A} f(\mathbf{x})$. Then,

$$\min_{\mathbf{x}\in A} f(\mathbf{x}) - \min_{\mathbf{x}\in A} g(\mathbf{x}) = f(\tilde{\mathbf{x}}) - \min_{\mathbf{x}\in A} g(\mathbf{x}) \geq f(\tilde{\mathbf{x}}) - g(\tilde{\mathbf{x}}) \geq \min_{\mathbf{x}\in A}\{f(\mathbf{x}) - g(\mathbf{x})\}. \tag{C.18}$$

(b) We show $\tilde{e}(\{\mathbf{y} \in \mathcal{Y}, x_\zeta = r\}) \geq e^{rc}(\{\mathbf{y} \in \mathcal{Y}, x_\zeta = c_k\})$ is true for all $\{\mathbf{y} \in \mathcal{Y} | \tilde{g}(\{\mathbf{y}, x_\zeta = r\}, \tilde{P}) = 0\}$ and $r \in C_k$, for each $k = 1, \dots, l$.

By definition of $e^{rc}_{\zeta j}$, i.e. $e^{rc}_{\zeta j}(c_k, s) = \min_{r\in C_k, (r,s)\notin \tilde{P}} \tilde{e}_{\zeta j}(r, s)$, we know $\tilde{e}_{\zeta j}(r, s) \geq e^{rc}_{\zeta j}(c_k, s)$ is true for all $r \in C_k$, $s \in R_j$ such that $(r, s) \notin \tilde{P}$, for each $j \in \Gamma(\zeta)$ and $k = 1, \dots, l$. Since $\tilde{e}(\mathbf{x})$ and $e^{rc}(\mathbf{x})$ are different only for (ζ, j) pairwise terms such that $j \in \Gamma(\zeta)$, the claim is true.

(c) We show part of the inequality in Lemma 5.

We know $e^{rc}_{\zeta j}(c_k, s, \tilde{P}) \geq e^{rc}_{\zeta j}(c_k, s, \phi)$ for all $s \in R_j$, $j \in \Gamma(\zeta)$, and $k = 1, \dots, l$ because $e^{rc}_{\zeta j}(c_k, s) = \min_{r\in C_k, (r,s)\notin \tilde{P}} \tilde{e}_{\zeta j}(r, s)$ while $e^{rc}_{\zeta j}(c_k, s, \phi) = \min_{r\in C_k} \tilde{e}_{\zeta j}(r, s)$. Since the rest energy terms are all same between $e^{rc}(\mathbf{x}, \tilde{P})$ and $e^{rc}(\mathbf{x}, \phi)$ by construction, we obtain $e^{rc}(\mathbf{x}, \tilde{P}) \geq e^{rc}(\mathbf{x}, \phi)$ for all $\mathbf{x} \in \mathcal{X}^{rc}$. Therefore,

$$\min_{\mathbf{x}\in \mathcal{X}^{rc}} e^{rc}(\mathbf{x}, \phi) \leq \min_{\mathbf{x}\in \mathcal{X}^{rc}} e^{rc}(\mathbf{x}, \tilde{P}) \leq \min_{\mathbf{x}\in \mathcal{L}(\mathcal{X}^{rc}, \tilde{P}^{rc})\}} e^{rc}(\mathbf{x}, \tilde{P}). \tag{C.19}$$

(d) We complete the inequality of Lemma 5 by using part (1a), (2a), and (2b):

If $\mathcal{L}(\mathcal{X}, \tilde{P})$ is empty, we regard $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x})$ as $+\infty$ since the subproblem will be pruned by any U; therefore, $\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x}) \geq \min_{\mathbf{x}\in\mathcal{L}(\mathcal{X}^{rc},\tilde{P}^{rc})} e^{rc}(\mathbf{x})$.

On the other hand, if $\mathcal{L}(\mathcal{X}, \tilde{P}) \neq \phi$,

$$\begin{aligned}
&\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} \tilde{e}(\mathbf{x}) - \min_{\mathbf{x}\in\mathcal{L}(\mathcal{X}^{rc},\tilde{P}^{rc})} e^{rc}(\mathbf{x}) \\
&= \min_{k=1,\ldots,l} \min_{r\in C_k} \min_{\{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})|x_\zeta=r\}} \tilde{e}(\mathbf{x}) - \min_{k=1,\ldots,l} \min_{\{\mathbf{x}\in\mathcal{L}(\mathcal{X}^{rc},\tilde{P}^{rc})|x_\zeta=c_k\}} e^{rc}(\mathbf{x}) \\
&\geq \min_{k=1,\ldots,l} \min_{r\in C_k} \left\{ \min_{\{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})|x_\zeta=r\}} \tilde{e}(\mathbf{x}) - \min_{\{\mathbf{x}\in\mathcal{L}(\mathcal{X}^{rc},\tilde{P}^{rc})|x_\zeta=c_k\}} e^{rc}(\mathbf{x}) \right\} \\
&= \min_{k=1,\ldots,l} \min_{r\in C_k} \left\{ \min_{\{\mathbf{y}\in\mathcal{Y}|\tilde{g}(\{\mathbf{y},x_\zeta=r\},\tilde{P})=0\}} \tilde{e}(\{\mathbf{y}, x_\zeta = r\}) \right. \\
&\qquad \left. - \min_{\{\mathbf{y}\in\mathcal{Y}|\tilde{g}(\{\mathbf{y},x_\zeta=c_k\},\tilde{P}^{rc})=0\}} e^{rc}(\{\mathbf{y}, x_\zeta = c_k\}) \right\} \\
&\geq \min_{k=1,\ldots,l} \min_{r\in C_k} \left\{ \min_{\{\mathbf{y}\in\mathcal{Y}|\tilde{g}(\{\mathbf{y},x_\zeta=r\},\tilde{P})=0\}} \tilde{e}(\{\mathbf{y}, x_\zeta = r\}) \right. \\
&\qquad \left. - \min_{\{\mathbf{y}\in\mathcal{Y}|\tilde{g}(\{\mathbf{y},x_\zeta=r\},\tilde{P})=0\}} e^{rc}(\{\mathbf{y}, x_\zeta = c_k\}) \right\} \\
&\geq \min_{k=1,\ldots,l} \min_{r\in C_k} \min_{\{\mathbf{y}\in\mathcal{Y}|\tilde{g}(\{\mathbf{y},x_\zeta=r\},\tilde{P})=0\}} \{\tilde{e}(\{\mathbf{y}, x_\zeta = r\}) - e^{rc}(\{\mathbf{y}, x_\zeta = c_k\})\} \\
&\geq 0.
\end{aligned}$$

C.7 Proof of Lemma 6

We will use the following inequality for arbitrary finite functions $f(\mathbf{x})$ and $g(\mathbf{x})$ both defined over some non-empty finite set A:

$$\min_{\mathbf{x}\in A} f(\mathbf{x}) - \min_{\mathbf{x}\in A} g(\mathbf{x}) \leq \max_{\mathbf{x}\in A}\{f(\mathbf{x}) - g(\mathbf{x})\}. \tag{C.20}$$

A proof for (C.20) is as follows: let $\tilde{\mathbf{x}} \in \arg\min_{\mathbf{x}\in A} g(\mathbf{x})$. Then,

$$\min_{\mathbf{x}} f(\mathbf{x}) - \min_{\mathbf{x}} g(\mathbf{x}) = \min_{\mathbf{x}} f(\mathbf{x}) - g(\tilde{\mathbf{x}}) \leq f(\tilde{\mathbf{x}}) - g(\tilde{\mathbf{x}}) \leq \max_{\mathbf{x}\in A}\{f(\mathbf{x}) - g(\mathbf{x})\}.$$

Now, if $\mathcal{L}(\mathcal{X}^{rc}, \tilde{P}^{rc})$ is empty, ΔOPT^{rc} is $-\infty$. If we let $e^{rc}(\mathbf{x})$ and $\mathcal{Y}$ defined same as in the proof of Lemma 5,

$$\begin{aligned}
\Delta OPT^{rc} &= \min_{\mathbf{x}\in\mathcal{X}} \tilde{e}(\mathbf{x}) - \min_{\{\mathbf{x}\in\mathcal{L}(\mathcal{X}^{rc},\tilde{P}^{rc})\}} e^{rc}(\mathbf{x}) \\
&= \min_{k=1,\dots,l} \min_{r\in C_k} \min_{\{\mathbf{x}\in\mathcal{X}|x_\zeta=r\}} \tilde{e}(\mathbf{x}) - \min_{k=1,\dots,l} \min_{\{\mathbf{x}\in\mathcal{L}(\mathcal{X}^{rc},\tilde{P}^{rc})|x_\zeta=c_k\}} e^{rc}(\mathbf{x}) \\
&\le \max_{k=1,\dots,l} \left\{ \min_{r\in C_k} \min_{\{\mathbf{x}\in\mathcal{X}|x_\zeta=r\}} \tilde{e}(\mathbf{x}) - \min_{\{\mathbf{x}\in\mathcal{L}(\mathcal{X}^{rc},\tilde{P}^{rc})|x_\zeta=c_k\}} e^{rc}(\mathbf{x}) \right\} \\
&\le \max_{k=1,\dots,l} \min_{r\in C_k} \left\{ \min_{\mathbf{y}\in\mathcal{Y}} \tilde{e}(\{\mathbf{y}, x_\zeta = r\}) \right. \\
&\qquad \left. - \min_{\{\mathbf{y}\in\mathcal{Y}|\tilde{g}(\{\mathbf{y},x_\zeta=c_k\},\tilde{P}^{rc})=0\}} e^{rc}(\{\mathbf{y}, x_\zeta = c_k\}) \right\} \\
&\le \max_{k=1,\dots,l} \min_{r\in C_k} \left\{ \min_{\{\mathbf{y}\in\mathcal{Y}|\tilde{g}(\{\mathbf{y},x_\zeta=c_k\},\tilde{P}^{rc})=0\}} \tilde{e}(\{\mathbf{y}, x_\zeta = r\}) \right. \\
&\qquad \left. - \min_{\{\mathbf{y}\in\mathcal{Y}|\tilde{g}(\{\mathbf{y},x_\zeta=c_k\},\tilde{P}^{rc})=0\}} e^{rc}(\{\mathbf{y}, x_\zeta = c_k\}) \right\} \qquad \text{(C.21)} \\
&\le \max_{k=1,\dots,l} \min_{r\in C_k} \max_{\{\mathbf{y}\in\mathcal{Y}|\tilde{g}(\{\mathbf{y},x_\zeta=c_k\},\tilde{P}^{rc})=0\}} \{\tilde{e}(\{\mathbf{y}, x_\zeta = r\}) - e^{rc}(\{\mathbf{y}, x_\zeta = c_k\})\} \\
&= \max_{k=1,\dots,l} \min_{r\in C_k} \max_{\{\mathbf{y}\in\mathcal{Y}|\tilde{g}(\{\mathbf{y},x_\zeta=c_k\},\tilde{P}^{rc})=0\}} \sum_{j\in\Gamma(\zeta)} \{\tilde{e}_{\zeta j}(r, y_j) - e^{rc}_{\zeta j}(c_k, y_j)\} \\
&\le \max_{k=1,\dots,l} \min_{r\in C_k} \sum_{j\in\Gamma(\zeta)} \max_{\{s\in R_j|(c_k,s)\notin\tilde{P}^{rc}\}} \{\tilde{e}_{\zeta j}(r, s) - e^{rc}_{\zeta j}(c_k, s)\} \\
&= U^{rc}_{\Delta OPT}(\tilde{P}) \\
&\le \max_{k=1,\dots,l} \min_{r\in C_k} \sum_{j\in\Gamma(\zeta)} \max_{s\in R_j} \{e_{\zeta j}(r, s) - e^{rc}_{\zeta j}(c_k, s, \phi)\} = U^{rc}_{\Delta OPT}(\phi). \qquad \text{(C.22)}
\end{aligned}$$

(C.21) holds by the fact $\mathcal{Y} \supset \{\mathbf{y} \in \mathcal{Y}|\tilde{g}(\{\mathbf{y}, x_\zeta = c_k\}, \tilde{P}^{rc}) = 0\}$. The last inequality (C.22) is true because $\{s \in R_j|(c_k, s) \notin \tilde{P}^{rc}\} \subset R_j$ and $e^{rc}_{\zeta j}(c_k, s, \tilde{P}) > e^{rc}_{\zeta j}(c_k, s, \phi)$ for all $s \in R_j$, $j \in \Gamma(\zeta)$, and $k = 1, \dots, l$.

C.8 Proof of Lemma 7

Let $\tilde{\mathbf{x}} \in \arg\min_{\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x})$. Then, we know $\hat{e}(\tilde{\mathbf{x}}) = e(\tilde{\mathbf{x}}) = \min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x})$. However, we have $\hat{e}(\mathbf{x}) \ge M \gg \hat{e}(\tilde{\mathbf{x}})$ for all $\mathbf{x} \in \mathcal{X}$ such that $\tilde{g}(\mathbf{x}, \tilde{P}) > 0$. Therefore, we obtain $\arg\min_{\mathbf{x}\in\mathcal{X}} \hat{e}(\mathbf{x}) \cap \{\mathbf{x} \in \mathcal{X}|\tilde{g}(\mathbf{x}, \tilde{P}) > 0\} = \phi$. Thus, $\arg\min_{\mathbf{x}\in\mathcal{X}} \hat{e}(\mathbf{x}) =$

$\arg\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} \hat{e}(\mathbf{x})$. Since $\hat{e}(\mathbf{x}) = e(\mathbf{x})$ for all $\mathbf{x}$ such that $\tilde{g}(\mathbf{x},\tilde{P}) = 0$, we obtain $\arg\min_{\mathbf{x}\in\mathcal{X}} \hat{e}(\mathbf{x}) = \arg\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} \hat{e}(\mathbf{x}) = \arg\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X},\tilde{P})} e(\mathbf{x})$.

C.9 Proof of Lemma 8

There are three kinds of operations that can change the pair-flags and the energy function: flagging, singles-elimination, and unification. For clarity, the claim is rewritten as follows by denoting n the count of operations performed in *DEE-gp*:

1. $\arg\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X}^{(n)},\tilde{P}^{(n)})} e^{(n)}(\mathbf{x}) = \arg\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X}^{(n)},\hat{P}^{(n)})} \hat{e}^{(n)}(\mathbf{x})$ if $\mathcal{L}(\mathcal{X},\tilde{P}) \neq \phi$, and $\mathcal{L}(\mathcal{X}^{(n)},\tilde{P}^{(n)}) = \phi$ if $\mathcal{L}(\mathcal{X},\tilde{P}) = \phi$.

2. for any $\mathbf{z} \in \arg\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X}^{(n)},\tilde{P}^{(n)})} e^{(n)}(\mathbf{x})$, we have $e^{(n)}(\mathbf{z}) = \hat{e}^{(n)}(\mathbf{z})$.

For each n, let $A^{(n)} \overset{def}{=} \arg\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X}^{(n)},\tilde{P}^{(n)})} e^{(n)}(\mathbf{x})$ and $B^{(n)} \overset{def}{=} \arg\min_{\mathbf{x}\in\mathcal{L}(\mathcal{X}^{(n)},\hat{P}^{(n)})} \hat{e}^{(n)}(\mathbf{x})$. We know $B^{(0)} \neq \phi$ since $\hat{P}^{(0)}$ is empty.

For $n = 0$, we have $\tilde{P}^{(0)} = \tilde{P}$ and $e^{(0)}(\mathbf{x}) = e(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$. Therefore, if $\mathcal{L}(\mathcal{X},\tilde{P}) = \phi$, then $\mathcal{L}(\mathcal{X}^{(n)},\tilde{P}^{(n)}) = \phi$. If $\mathcal{L}(\mathcal{X},\tilde{P}) \neq \phi$, then $A^{(0)} = B^{(0)}$ because $\hat{P}^{(0)}$ is empty and by Lemma 7. On the other hand, for any $\mathbf{z} \in A^{(0)}$, we know $\tilde{g}(\mathbf{z},\tilde{P}) = 0$. Therefore, $\hat{e}^{(0)}(\mathbf{z}) = e^{(0)}(\mathbf{z})$.

Now we show the claim is true after $(n+1)$th operation if it is true after nth operation. It is straightforward to see that $\mathcal{L}(\mathcal{X}^{(n)},\tilde{P}^{(n+1)}) = \phi$ if $\mathcal{L}(\mathcal{X}^{(n)},\tilde{P}^{(n)}) = \phi$ because each operation eliminates a set of conformations (possibly an empty set). Therefore, we assume $\mathcal{L}(\mathcal{X},\tilde{P}) \neq \phi$ in what follows.

1. Flagging: Flagging does not modify $\tilde{P}^{(n)}$, $e^{(n)}(\mathbf{x})$, or $\hat{e}^{(n)}(\mathbf{x})$. Therefore, $A^{(n+1)} = A^{(n)}$. However, we have $\hat{P}^{(n)} \subset \hat{P}^{(n+1)}$. Since $\mathcal{L}(\mathcal{X}^{(n)},\hat{P}^{(n)}) \supset \mathcal{L}(\mathcal{X}^{(n)},\hat{P}^{(n+1)})$ and $\hat{e}^{(n+1)}(\mathbf{x}) = \hat{e}^{(n)}(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$, we have $B^{(n)} \supset B^{(n+1)}$. Because DEE does not flag GMEC rotamer-pairs, we know any $\mathbf{z} \in B^{(n)}$ satisfies $\tilde{g}(\mathbf{z},\hat{P}^{(n+1)}) = 0$ and is also $\mathbf{z} \in B^{(n+1)}$. Therefore, we obtain $B^{(n+1)} = B^{(n)} = A^{(n)} = A^{(n+1)}$.

   Let $\mathbf{z} \in A^{(n+1)}$. Then, we also have $\mathbf{z} \in A^{(n)} = B^{(n)}$. Since the energy functions are not altered, we have $e^{(n+1)}(\mathbf{z}) = e^{(n)}(\mathbf{z}) = \hat{e}^{(n)}(\mathbf{z}) = \hat{e}^{(n+1)}(\mathbf{z})$.

2. Singles-elimination: Singles-elimination only eliminates conformations containing flagged rotamer-pairs. From $A^{(n)} = B^{(n)}$, for any $\mathbf{z} \in A^{(n)}$, we have $\tilde{g}(\mathbf{z}, \tilde{P}^{(n)}) = \tilde{g}(\mathbf{z}, \hat{P}^{(n)}) = 0$. Therefore, $\mathbf{z}$ contains no flagged rotamer-pair, and will not be eliminated by singles-elimination. So we have $A^{(n)} \subset \mathcal{L}(\mathcal{X}^{(n)}, \tilde{P}^{(n+1)})$. On the other hand, the energy terms are not changed by singles-elimination but only the conformation space is reduced. Therefore, we have $A^{(n+1)} = A^{(n)}$.

   Similarly, we can also obtain $B^{(n+1)} = B^{(n)}$. Therefore, $A^{(n+1)} = B^{(n+1)}$.

   We obtain $e^{(n+1)}(\mathbf{z}) = \hat{e}^{(n+1)}(\mathbf{z})$ for $\mathbf{z} \in A^{(n+1)}$ by the same argument used in flagging.

3. Unification: For any $\mathbf{z}' \in \mathcal{L}(\mathcal{X}^{(n)}, \tilde{P}^{(n+1)})$, there exists $\mathbf{z} \in \mathcal{L}(\mathcal{X}^{(n)}, \tilde{P}^{(n)})$ by a mapping such that

$$z'_i = z_i \text{ for } i \neq [\zeta : \eta], \tag{C.23}$$

$$z'_{[\zeta:\eta]} = [z_\zeta : z_\eta]. \tag{C.24}$$

   In addition, we also have $e^{(n+1)}(\mathbf{z}') = e^{(n)}(\mathbf{z})$ for such $\mathbf{z}$ and $\mathbf{z}'$. On the other hand, any $\mathbf{z} \in A^{(n)}$ have a mapping $\mathbf{z}' \in A^{(n+1)}$ because such $\mathbf{z}$ satisfies $\tilde{g}(\mathbf{z}, \tilde{P}^{(n)}) = \tilde{g}(\mathbf{z}, \hat{P}^{(n)}) = 0$ from $A^{(n)} = B^{(n)}$ and therefore none of the rotamer-pair in $\mathbf{z}'$ will be in $\tilde{P}^{(n+1)}$. So there exists a one-to-one mapping between elements of $A^{(n)}$ and $A^{(n+1)}$.

   A similar argument can be made to claim there exists a one-to-one mapping between $B^{(n)}$ and $B^{(n+1)}$. Then, from $A^{(n)} = B^{(n)}$ and because the same type of mapping is used for $A^{(n)} \leftrightarrow A^{(n+1)}$ and $B^{(n)} \leftrightarrow B^{(n+1)}$, we obtain $A^{(n+1)} = B^{(n+1)}$.

   Let $\mathbf{z}' \in A^{(n+1)}$ is a mapping of $\mathbf{z} \in A^{(n)}$ by (C.23) and (C.24). Then, we obtain $e^{(n+1)}(\mathbf{z}') = e^{(n)}(\mathbf{z}) = \hat{e}^{(n)}(\mathbf{z}) = \hat{e}^{(n+1)}(\mathbf{z}')$.

C.10 Proof of Corollary 2

If $\mathcal{L}(\mathcal{X}, \tilde{P}) = \phi$, we know $\mathcal{L}(\mathcal{X}^{(n)}, \tilde{P}^{(n)}) = \phi$ for every n from Lemma 8 (see the proof of Lemma 8 for the notation). Therefore, $\mathcal{L}(\mathcal{X}, \tilde{P}') = \mathcal{L}(\mathcal{X}^{(n)}, \tilde{P}^{(n)} \cup \hat{P}^{(n)}) \subset \mathcal{L}(\mathcal{X}^{(n)}, \tilde{P}^{(n)}) = \phi$.

If $\mathcal{L}(\mathcal{X}, \tilde{P}) \neq \phi$, since DEE only eliminates non-GMEC rotamers and rotamer-pairs, we have

$$\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P})} e(\mathbf{x}) = \min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}^{(n)}, \hat{P}^{(n)})} \hat{e}^{(n)}(\mathbf{x}), \tag{C.25}$$

for every n. By Lemma 8.1, we have $A^{(n)} = B^{(n)}$. So for every $\mathbf{z} \in A^{(n)}$, we have $\tilde{g}(\mathbf{z}, \tilde{P}^{(n)}) = \tilde{g}(\mathbf{z}, \hat{P}^{(n)}) = 0$. Let N be the the last count of operations in *DEE-gp* and $\mathbf{z}^{(N)} \in A^{(N)}$. The last step of *DEE-gp* ($\tilde{P}' \leftarrow \tilde{P}^{(N)} \cup \hat{P}^{(N)}$) gives $\tilde{g}(\mathbf{z}, \tilde{P}') = \tilde{g}(\mathbf{z}, \tilde{P}^{(N)}) \vee \tilde{g}(\mathbf{z}, \hat{P}^{(N)}) = 0$. Then, from

$$\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}^{(n)}, \tilde{P}^{(n)})} e^{(N)}(\mathbf{x}) \leq \min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}^{(n)}, \tilde{P}^{(N)} \cup \hat{P}^{(N)})} e^{(N)}(\mathbf{x}) = \min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P}')} e'(\mathbf{x}) \leq e^{(N)}(\mathbf{z}^{(N)}),$$

we obtain

$$\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P}')} e'(\mathbf{x}) = e^{(N)}(\mathbf{z}^{(N)}). \tag{C.26}$$

Finally, from $e^{(N)}(\mathbf{z}^{(N)}) = \hat{e}^{(N)}(\mathbf{z}^{(N)})$ by Lemma 8.2, and the equations (C.25) and (C.26), we obtain $\min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P}')} e'(\mathbf{x}) = \min_{\mathbf{x} \in \mathcal{L}(\mathcal{X}, \tilde{P})} e(\mathbf{x})$.

C.11 Proof of Lemma 9

Any $\mu \in \mathcal{M}(\mathcal{G}; \mathcal{G}^h)$ satisfies

$$\sum_{r \in R_i} \mu_{i_r} = \sum_{r \in R_i} p_i(x_i) = 1, \tag{C.27}$$

for all $i \in \mathcal{V}(G^h)$, and

$$\sum_{s \in R_j} \mu_{i_r j_s} = \sum_{s \in R_j} p(x_i = r, x_j) = p(x_i = r) = \mu_{i_r}, \tag{C.28}$$

for all $(i,j) \in \mathcal{E}(G^h)$ because $p(\mathbf{x})$ is a distribution. From $\mathcal{V}(T^l) \subset \mathcal{V}(G^h)$ and $\mathcal{E}(T^l) \subset \mathcal{E}(G^h)$, we have $\mathcal{M}(\mathcal{G};\mathcal{G}^h) \subset \text{LOCAL}(\mathcal{G};T^l)$ for all l, and therefore $\mathcal{M}(\mathcal{G};\mathcal{G}^h) \subset \bigcap_l \text{LOCAL}(\mathcal{G};T^l)$. On the other hand, if we let

$$\pi(x_c) = \sum_{\{x' | x'_C = x_C\}} p(\mathbf{x}'), \tag{C.29}$$

then

$$\sum_{\{x'_C | (x'_i, x'_j) = (r,s)\}} \pi(x'_c) = \sum_{\{x'_C | (x'_i, x'_j) = (r,s)\}} \sum_{\{x'' | x''_C = x'_C\}} p(\mathbf{x}'') = p_{ij}(x_i = r, x_j = s) = \mu_{i_r j_s}, \tag{C.30}$$

for all $(i,j) \in \mathcal{E}(C) \cap \mathcal{E}(G^h)$, and therefore, $\mathcal{M}(\mathcal{G};\mathcal{G}^h) \subset \mathcal{C}(\mathcal{G};\mathcal{G}^h, C)$.

C.12 Proof of Lemma 10

We use the index set notation $\mathcal{I}(\mathcal{G})$ defined as:

$$\mathcal{I}(\mathcal{G}) = \left\{ \bigcup_{i \in \mathcal{V}} \{ i_r \mid r \in R_i \} \right\} \cup \left\{ \bigcup_{(i,j) \in \mathcal{E}} \{ i_r j_s \mid (r,s) \in R_i \times R_j \} \right\}. \tag{C.31}$$

We can show the following equalities are true:

$$\min_{x^h \in \mathcal{F}^h} \theta^h \cdot x^h \overset{(a)}{=} \min_{x \in \mathcal{F}} \phi(\theta^h) \cdot x \overset{(b)}{=} \min_{\tau \in \mathcal{M}(\mathcal{G})} \phi(\theta^h) \cdot \tau \overset{(c)}{=} \min_{\tau \in \mathcal{M}(\mathcal{G};\mathcal{G}^h)} \phi(\theta^h) \cdot \tau, \tag{C.32}$$

where equality (a) is from the definition of $\phi(\cdot)$, and equality (b) is Lemma 1 of Wainwright et al. [2005]. Equality (c) follows from $\phi(\theta^h) \cdot \mu = \phi(\theta^h) \cdot \mu'$ for any $\mu \in \mathcal{M}(\mathcal{G})$ and $\mu' \in \mathcal{M}(\mathcal{G};\mathcal{G}^h)$ such that $\mu_\alpha = \mu'_\alpha$ for all $\alpha \in \mathcal{I}(\mathcal{G}^h)$, because $\phi(\theta^h)_\alpha = 0$ for all $\alpha \in \mathcal{I}(\mathcal{G}) \backslash \mathcal{I}(\mathcal{G}^h)$ and

$$\{\mu_\alpha \ \forall \alpha \in \mathcal{I}(\mathcal{G}^h) \mid \mu \in \mathcal{M}(\mathcal{G})\} = \{\mu'_\alpha \ \forall \alpha \in \mathcal{I}(\mathcal{G}^h) \mid \mu' \in \mathcal{M}(\mathcal{G};\mathcal{G}^h)\}. \tag{C.33}$$

From (C.32), if $\tau \in \mathcal{M}(\mathcal{G};\mathcal{G}^h)$, then $\min_{x^h \in \mathcal{F}^h} \theta^h \cdot x^h - \tau \cdot \phi(\theta^h) \leq 0$ for all $\theta^h \in \mathbb{R}^{d(h)}$, with equality for $\theta^h = 0$. Therefore,

$$\sup_{\theta^h \in \mathbb{R}^{d(G^h)}} \left\{ \min_{x^h \in \mathcal{F}^h} \theta^h \cdot x^h - \tau \cdot \phi(\{\theta^h\}) \right\} = 0, \tag{C.34}$$

if $\tau \in \mathcal{M}(\mathcal{G}; \mathcal{G}^h)$.

If $\tau \notin \mathcal{M}(\mathcal{G}; \mathcal{G}^h)$, then by the separating hyperplane theorem Bertsekas [1999], there exist some vector $\zeta \in \mathbb{R}^{d(\mathcal{G})}$ and a scalar η such that

$$\zeta \cdot \tau > \eta, \tag{C.35}$$

and

$$\zeta \cdot \mu \leq \eta \text{ for all } \mu \in \mathcal{M}(\mathcal{G}; \mathcal{G}^h). \tag{C.36}$$

We know that μ_α is not constrained for all $\alpha \notin \mathcal{I}(\mathcal{G}) \backslash \mathcal{I}(\mathcal{G}^h)$ and $\mu \in \mathcal{M}(\mathcal{G}, \mathcal{G}^h)$. Therefore, $\zeta_\alpha = 0$ for all $\alpha \notin \mathcal{I}(\mathcal{G}) \backslash \mathcal{I}(\mathcal{G}^h)$; otherwise, $\zeta_\alpha \cdot \mu_\alpha$ can be tended to $\pm\infty$ as we please, and therefore, (C.35) and (C.36) cannot be true. Therefore, there exists some $\zeta(\mathcal{G}^h)$ such that $\zeta = \phi(\zeta(\mathcal{G}^h))$. Then, from (C.36) and Lemma 1 of Wainwright et al. [2005], we have

$$\min_{\mu \in \mathcal{M}(\mathcal{G}; \mathcal{G}^h)} \zeta \cdot \mu = \min_{\mu' \in \mathcal{M}(\mathcal{G}^h)} \zeta(\mathcal{G}^h) \cdot \mu' = \min_{x^h \in \mathcal{F}^h} \zeta(G^h) \cdot x^h \leq \eta. \tag{C.37}$$

From (C.36) and (C.37),

$$\phi(\zeta(\mathcal{G}^h)) \cdot \tau - \min_{x^h \in \mathcal{F}^h} \zeta(G^h) \cdot x^h \geq \phi(\zeta(\mathcal{G}^h)) \cdot \tau - \eta > 0. \tag{C.38}$$

Finally, since we can multiply both sides of (C.35) or (C.36) with any positive number, we can scale $\phi(\zeta(\mathcal{G}^h))$ and η with any large positive number as we please. By tending this multiplier to $+\infty$, $\phi(\zeta(\mathcal{G}^h)) \cdot \tau - \eta$ also tends to $+\infty$. Therefore,

$$\sup_{\theta^h \in \mathbb{R}^{d(G^h)}} \left\{ \min_{x^h \in \mathcal{F}^h} \theta^h \cdot x^h - \tau \cdot \phi(\{\theta^h\}) \right\} = +\infty, \tag{C.39}$$

if $\tau \notin \mathcal{M}(\mathcal{G}; \mathcal{G}^h)$.

C.13 Proof of Theorem 3

We are going to take problem (6.27) as the primal problem and take the dual of it to obtain problem (6.31) as the dual problem.

We obtain the following Lagrangian function by relaxing the constraints on $\{\theta^h\}$ of (6.27):

$$L(\{\theta^h\}, \tau) = \sum_h \min_{x^h \in \mathcal{F}^h} \theta^h \cdot x^h + \tau \cdot \{c - \phi(\{\theta^h\})\} \tag{C.40}$$

Then, we obtain the following dual function:

$$\begin{aligned} Q(\tau) &= \sup_{\{\theta^h\} \in \mathbb{R}^{\tilde{d}}} L(\{\theta^h\}, \tau) && \text{(C.41)} \\ &= \sum_h \sup_{\theta^h \in \mathbb{R}^{d(G^h)}} \left\{ \min_{x^h \in \mathcal{F}^h} \theta^h \cdot x^h - \tau \cdot \phi(\theta^h) \right\} + \tau \cdot c && \text{(C.42)} \end{aligned}$$

By Lemma 10, the dual function (C.42) is simplified to

$$Q(\tau) = \begin{cases} \tau \cdot c & \text{if } \tau \in \mathcal{M}(G; G^h), \\ +\infty & \text{otherwise.} \end{cases} \tag{C.43}$$

Finally, since the dual function $Q(\tau)$ is to be minimized, we obtain the following as the dual problem of (6.27):

$$\begin{aligned} &\text{minimize} && \tau \cdot c = \tau \cdot \{\{E_{i_r}\}, \{E_{i_r j_s}\}\} \\ &\text{subject to} && \tau \in \textstyle\bigcap_h \mathcal{M}(\mathcal{G}; \mathcal{G}^h). \end{aligned} \tag{C.44}$$

C.14 Proof of Lemma 11

Let $\{T^{h,l}\}$ be the tree decomposition for G^h for each h. Then, by Lemma 9, we have

$$\begin{aligned} \bigcap_h \mathcal{M}(\mathcal{G}; \mathcal{G}^h) &\subset \bigcap_h \bigcap_l \text{LOCAL}(\mathcal{G}; T^{h,l}) \cap \mathcal{C}(\mathcal{G}^h; C^h) \\ &= \text{LOCAL}(\mathcal{G}) \bigcap_h \mathcal{C}(\mathcal{G}^h; C^h) \subset \text{LOCAL}(\mathcal{G}). \end{aligned}$$

C.15 Proof of Lemma 12

Let $i, j \in D(\Upsilon)$, $i \neq j$.

1. Suppose there exists $v \in \mathcal{V} \backslash X_s$ such that $v \in X_{T(i)}$ and $v \in X_{T(j)}$. Then, by Definition 1-3, v should be also in X_s. This contradicts the assumption.

2. Suppose there exists $(u, v) \in \mathcal{E}$ such that $u \in X_{T(i)} \backslash X_s$ and $v \in X_{T(j)} \backslash X_s$. By Definition 1-2, there exists $k \in I$, $k \neq \Upsilon$ such that $u \in X_k$ and $v \in X_k$. Then, i and j are both neighbors of k, therefore, i, j, k, and Υ induce a cycle. This contradicts Definition 1-3.

C.16 Proof of Lemma 13

For each $v \in V$, there exists a unique $i \in I$ such that $\mathit{depth}(i) = \min_{\{j \in I | v \in X_j\}} \mathit{depth}(j)$; if there exist $i, k \in I, i \neq k$ such that $\mathit{depth}(i) = \mathit{depth}(k) = \min_{\{j \in I | v \in X_j\}} \mathit{depth}(j)$, then $i \neq k \neq \Upsilon$ because *Upsilon* is the only node with depth 1. In addition, $v \in X_i$ and $v \in X_k$. Therefore, there exist $l = \mathit{pa}(i) = \mathit{pa}(k)$ and $v \in X_l$ by the definition of tree decomposition. This contradicts the assumption.

Therefore, $S(i) \cap S(j) = \phi$ for all $i, j \in I, i \neq j$. In addition, for all $v \in V$, there exists some $i \in I$ such that $v \in X_i$ by the definition of tree decomposition. Therefore, $\bigcup_{i \in I} S(i) = \mathcal{V}$.

The second part for edges can be shown similarly.

C.17 Proof of Lemma 14

We can decompose $f(\mathbf{x})$ as

$$f(\mathbf{x}) = f^{\Upsilon}(\mathbf{x}^{\Upsilon}) + \sum_{i \in D(\Upsilon)} f^{T(i)}(\mathbf{x}^{T(i)}) \tag{C.45}$$

Then,

$$\begin{aligned}\min_{\{\tilde{\mathbf{x}}|\tilde{\mathbf{x}}^{\Upsilon}=\mathbf{x}^{\Upsilon}\}} f(\mathbf{x}) &= f^{\Upsilon}(\mathbf{x}^{\Upsilon}) + \min_{\{\tilde{\mathbf{x}}|\tilde{\mathbf{x}}^{\Upsilon}=\mathbf{x}^{\Upsilon}\}} \sum_{i\in D(\Upsilon)} f^{T(i)}(\tilde{\mathbf{x}}^{T(i)}) \\ &= f^{\Upsilon}(\mathbf{x}^{\Upsilon}) + \sum_{i\in D(\Upsilon)} \min_{\{\tilde{\mathbf{x}}^{T(i)}|\tilde{\mathbf{x}}^{\Upsilon\cap i}=\mathbf{x}^{\Upsilon\cap i}\}} f^{T(i)}(\tilde{\mathbf{x}}^{T(i)}), \qquad \text{(C.46)}\end{aligned}$$

where the second equality of (C.46) follows because:

1. by Lemma 1, $X_{T(i)\backslash\Upsilon} \cap X_{T(j)\backslash\Upsilon} = \phi$, for any $i, j \in D(\Upsilon)$;
2. by Lemma 2, there exists no edge $(u, v) \in \mathcal{E}$ such that $u \in X_{T(i)\backslash\Upsilon}$ and $v \in X_{T(j)\backslash\Upsilon}$, for some $i, j \in D(\Upsilon)$, $i \neq j$;
3. for all $v \in \mathcal{V}$ such that $v \in X_{T(i)}$ and $v \in X_{T(j)}$ for some $i, j \in D(\Upsilon)$, $i \neq j$, we also have $v \in X_{\Upsilon}$.

Bibliography

Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

E. Althaus, O. Kohlbacher, H.-P. Lenhof, and P. Müller. A combinatorial approach to protein docking with flexible side-chains. *J. Comput. Biol.*, 9:597–612, 2002.

M. D. Altman. *Computational ligand design and analysis in protein complexes using inverse methods, combinatorial search, and accurate solvation modeling.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, U.S.A., 2006.

X. I. Ambroggio and B. Kuhlman. Computational design of a single amino acid sequence that can switch between two distinct protein folds. *J. Am. Chem. Soc.*, 128:1154–1161, 2006.

S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal of Matrix Analysis and Applications*, 8:277–284, 1987.

D. Baker and A. Šali. Protein structure prediction and structural genomics. *Science*, 294(5540):93–96, 2001.

E. Balas and N. Christofides. A restricted lagrangian approach to the traveling salesman problem. *Mathematical Programming*, 21:19–46, 1981.

E. P. Baldwin, O. Hajiseyedjavadi, W. A. Baase, and B. W. Matthews. The role of backbone flexibility in the accommodation of variants that repack the core of t4 lysozyme. *Science*, 262:1715–1718, 1993.

F. Barahona and R. Anbil. The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming*, 87(3):385–399, May 2000.

F. Barahona and A. R. Mahjoub. On the cut polytope. *Mathematical Programming*, 36:157–173, 1986.

R. J. Baxter. *Exactly solved models in statistical mechanics.* Academic Press, New York, 1982.

M. Bayati, D. Shah, and M. Sharma. Maximum weight matching via max-product belief propagation. In *Proceedings of International Symposium on Information Theory, 2005*, 2005.

Sergio Benedetto, Guido Montorsi, Dariush Divsalar, and Fabrizio Pollara. Soft-output decoding algorithms in iterative decoding of turbo codes. Technical Report JPL TDA Progress Report 42-124, Jet Propulsion Laboratory, Pasadena, CA, 1996.

D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA, 1999.

T. N. Bhat, G. A. Bentley, G. Boulot, M. I. Greene, D. Tello, W. Dallacqua, H. Souchon, F. P. Schwarz, R. A. Mariuzza, and R. J. Poljak. Bound water molecules and conformational stabilization help mediate an antigen–antibody association. *Proc. Natl. Acad. Sci. U.S.A.*, 91:1089–1093, 1994.

H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal of Computing*, 25:1305–1317, 1996.

B. C. Braden, H. Souchon, J. L. Eisele, G. A. Bentley, T. N. Bhat, J. Navaza, and R. J. Poljak. Three-dimensional structures of the free and the antigen-complexed Fab from monoclonal anti-lysozyme antibody D44.1. *J. Mol. Biol.*, 243(4):767–81, 1994.

H. L. Brodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and minimum elimination tree height. *J. Algorithms*, 18:238–255, 1995.

H. L. Brodlaender, F. V. Fomin, A. M. C. A. Koster, D. Kratsch, and D. M. Thilikos. On exact algorithms for treewidth. In Y. Azar and T. Erlebach, editors, *Proc. 14th Annual European Symp. Algorithms, ESA 2006*, volume 4168 of *LNCS*, pages 672–683. Springer, 2006.

B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. *J. Comput. Chem.*, 4:187–217, 1983.

V. F. Cavalcante, C. C. de Souza, and A. Lucena. A relax-and-cut algorithm for the set partitioning problem. *Computers & Operations Research*, 35:1963–1981, 2008.

B. S. Chevalier, T. Kortemme, M. S. Chadsey, D. Baker, R. J. Monnat, and B. L. Stoddard. Design, activity, and structure of a highly specific artificial endonuclease. *Mol. Cell*, 10:895–905, 2002.

C. Chothia and A. M. Lesk. The relation between the divergence of sequence and structure in proteins. *EMBO J.*, 5(4):823–826, 1986.

W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz, Jr., 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. *J. Am. Chem. Soc.*, 117:5179–5197, 1995.

R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegehalter. Springer-Verlag, New York, 1999.

B. I. Dahiyat and S. L. Mayo. Protein design automation. *Protein Sci.*, 5:895–903, 1996.

B. I. Dahiyat and S. L. Mayo. De novo protein design: fully automated sequence selection. *Science*, 278:82–87, 1997.

B. I. Dahiyat, D. B. Gordon, and S. L. Mayo. Automated design of the surface positions of protein helices. *Protein Sci.*, 6:1333–1337, 1997.

G. Dantas, B. Kuhlman, D. Callender, M. Wong, , and D. Baker. A large scale test of computational protein design: folding and stability of nine completely redesigned globular proteins. *J. Mol. Biol.*, 332:449–460, 2003.

A. P. Dawid. Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, 2:25–36, 1992.

W. F. DeGrado, Z. R. Wasserman, and J. D. Lear. Protein design, a minimalist approach. *Science*, 243:622–628, 1989.

J. R. Desjarlais and T. M. Handel. De novo design of the hydrophobic cores of proteins. *Protein Sci.*, 4:2006–2018, 1995.

J. R. Desjarlais and T. M. Handel. Side-chain and backbone flexibility in protein core design. *J. Mol. Biol.*, 289:305–318, 1999.

J. Desmet, M. De Maeyer, B. Hazes, and I. Lasters. The dead-end elimination theorem and its use in protein side-chain positioning. *Nature*, 356:539–542, 1992.

R. I. Dima, J. R. Banavar, and A. Maritan. Scoring functions in protein folding and design. *Protein Sci.*, 9:812–819, 2000.

K. E. Drexler. Molecular engineering: an approach to the development of general capabilities for molecular manipulation. *Proc. Natl. Acad. Sci. U.S.A.*, 78:5275–5278, 1981.

R. L. Dunbrack. Rotamer libraries in the 21(st) century. *Curr. Opin. Struct. Biol.*, 12(4):431–440, 2002.

R. L. Dunbrack and M. Karplus. Backbone-dependent rotamer library for proteins. *J. Mol. Biol.*, 230:543–574, 1993.

M. A. Dwyer, L. L. Looger, and H. W. Hellinga. Computational design of a biologically active enzyme. *Science*, 304:1967–1971, 2004.

J. Eckstein, C. A. Phillips, and W. E. Hart. Pico: an object oriented framework form parallel branch and bound. Technical report, RUTCOR, 2001.

O. Eriksson, Y. Zhou, and A. Elofsson. Side chain-positioning as an integer programming problem. In *Proceedings of WABI 2001*, volume 2149 of *LNCS*, pages 128–141. Springer, 2001.

L. Escudero, M. Guignard, and K. Malik. A lagrangian relax and cut approach for the sequential ordering with precedence constraints. *Annals of Operations Research*, 50:219–237, 1994.

U. Feige, M. Hajiaghayi, and J. R. Lee. Improved approximation algorithms for minimum-weight vertex separators. In *Proc. 37th Annual Symp. Theory of Computing, STOC 2005*, pages 563–572. ACM Press, 2005.

F. V. Fomin, D. Kratsch, and I. Todinca. Exact (exponential) algorithms for treewidth and minimum fill-in. In J. Diaz, J. Karhumaki, A. Lepisto, and D. Sanella, editors, *Proc. 31st Int. Colloquium on Automata, Languages and Programming, ICALP 2004*, volume 3124 of *LNCS*, pages 568–580. Springer, 2004.

R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Information Processing Letters*, 12:133–137, 1981.

William T. Freeman and Egon C. Pasztor. Learning to estimate scenes from images. *International Journal of Computer Vision*, 40:25–47, 2000.

William T. Freeman and Yair Weiss. On the fixed points of the max-product algorithm. Technical Report TR-99-39, MERL, 2000.

Francesca Fumero. A modified subgradient algorithm for lagrangean relaxation. *Computers & Operations Research*, 28:33–52, 2001.

I. Georgiev, R. H. Lilien, and B. R. Donald. Improved pruning algorithms and divide-and-conquer strategies for dead-end elimination, with application to protein design. *Bioinformatics*, 22(14):174–183, 2006.

A. Globerson and T. Jaakkola. Fixing max-product: Convergent message passing algorithms for map lp-relaxations. *Advances in Neural Information Processing Systems (NIPS) 21*, 2007a.

A. Globerson and T. Jaakkola. Approximate inference using planar graph decomposition. *Advances in Neural Information Processing Systems (NIPS) 20*, 2007b.

A. Godzik, A. Kolinski, and J. Skolnick. De novo and inverse folding predictions of protein structure and dynamics. *J. Comput. Aided Mol. Des.*, 7(4):397–438, 1993.

R. F. Goldstein. Efficient rotamer elimination applied to protein side-chains and related spin glasses. *Biophys. J.*, 66:1335–1340, 1994.

D. B. Gordon and S. L. Mayo. Branch-and-terminate: a combinatorial optimization algorithm for protein design. *Structure*, 7(9):1089–1098, 1999.

D. B. Gordon and S. L. Mayo. Radical performance enhancements for combinatorial optimization algorithms based on the dead-end elimination theorem. *J. Comput. Chem.*, 13:1505–1514, 1998.

D. B. Gordon, G. K. Hom, S. L. Mayo, and N. A. Pierce. Exact rotamer optimization for protein design. *J. Comput. Chem.*, 24:232–243, 2003.

M. Guignard. Efficient cuts in lagrangean 'relax-and-cut' schemes. *European Journal of Operational Research*, 105:216–223, 1998.

T. A. Halgren. Merck molecular force field. i. basis, form, scope, parameterization, and performance of mmff94. *J. Comput. Chem.*, 17:490–519, 1996.

P. B. Harbury, B. Tidor, and P. S. Kim. Repacking protein cores with backbone freedom: structure prediction for coiled coils. *Proc. Natl. Aacd. Sci. U.S.A.*, 92: 8408–8412, 1995.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cyber.*, 2:100–107, 1968.

H. W. Hellinga and F. M. Richards. Optimal sequence selection in proteins of known structure by simulated evolution. *Proc. Natl. Acad. Sci. U.S.A.*, 91:5803–5807, 1994.

Olivia Hunting, Ulrich Faigle, and Walter Kern. A lagrangian relaxation approach to the edge-weighted clique problem. *European Journal of Operations Research*, 131 (1):119–131, 2001.

X. Jiang, H. Farid, E. Pistor, and R. Farid. A new approach to the design of uniquely folded thermally stable proteins. *Protein Sci.*, 9:403–416, 2000.

J. K. Johnson, D. M. Malioutov, and A. S. Willsky. Lagrangian relaxation for map estimation in graphical models. In *Allerton Conference on Communication, Control, and Computing*, 2007.

D. T. Jones. De novo protein design using pairwise potentials and a genetic algorithm. *Protein Sci.*, 3:567–574, 1994.

W. L. Jorgensen and J. Tirado-Rives. Monte Carlo vs molecular dynamics for conformational sampling. *J. Phys. Chem.*, 100(34):14508–14513, 1996.

C. Kingsford, B. Chazelle, and M. Singh. Solving and analyzing side-chain positioning problems using linear and integer programming. *Bioinformatics*, 21(7):1028–1036, 2005.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

U. Kjaerulff. Triangulation of graphs: algorithms giving small total state space. Technical report, University of Aalborg, 1990.

P. Koehl and M. Delarue. Application of a self-consistent mean field theory to predict protein side-chains conformation and estimate their conformational entropy. *J. Mol. Biol.*, 239:249–275, 1994.

P. Koehl and M. Delarue. Mean-field minimization methods for biological macromolecules. *Curr. Opin. Struct. Biol.*, 6:222–226, 1996.

V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Trans. Pattern Anal.*, 28:1568–1583, 2006.

V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *IEEE Trans. Pattern Anal.*, 26:147–159, 2004.

N. Komodakis, N. Paragios, and G. Tziritas. Mrf optimization via dual decomposition: message-passing revisited. In *IEEE Conference in Computer Vision and Pattern Recognition (ICCV)*, 2007.

A. Korkegian, M. E. Black, D. Baker, and B. L. Stoddard. Computational thermostabilization of an enzyme. *Science*, 308:857–860, 2005.

A. M. C. A. Koster, Stan P. M. van Hoesel, and Antoon W. J. Kolen. The partial constraint satisfaction problem: Facets and lifting theorems. *Operations research letters*, 23(3-5):89–97, 1998.

A. M. C. A. Koster, Stan P. M. van Hoesel, and Antoon W. J. Kolen. Lower bounds for minimum interference frequency assignment problems. Technical Report RM 99/026, Maastricht University, 1999.

C. M. Kraemer-Pecore, J. T. Lecomte, and J. R. Desjarlais. A de novo redesign of the ww domain. *Protein Sci.*, 12:2194–2205, 2003.

B. Kuhlman and D. Baker. Exploring folding free energy landscapes using computational protein design. *Curr. Opin. Struct. Biol.*, 14:89–95, 2004.

B. Kuhlman, G. Dantas, G. C. Ireton, G. Varani, B. L. Stoddard, and D. Baker. Design of a novel globular protein fold with atomic-level accuracy. *Science*, 302: 1364–1368, 2003.

S. M. Larson, J. L. England, J. R. Desjarlais, and V. S. Pande. Thoroughly sampling sequence space: large-scale protein design of structural ensembles. *Protein Sci.*, 11: 2804–2813, 2002.

Torbjörn Larsson, Michael Patriksson, and Ann-Brith Strömberg. On the convergence of conditional ϵ-subgradient methods for convex programs and convex-concave saddle-point problems. *European Journal of Operational Research*, 151:461–473, 2003.

I. Lasters, M. De Maeyer, and J. Desmet. Enhanced dead-end elimination in the search for the global minimum energy conformation of a collection of protein side chains. *Protein Eng.*, 8:815–822, 1995.

G. A. Lazar, J. R. Desjarlais, and T. M. Handle. De novo design of the hydrophobic core of ubiquitin. *Protein Sci.*, 6:1167–1178, 1997.

A. R. Leach and A. P. Lemon. Exploring the conformational space of protein side chains using dead-end elimination and the A^* algorithm. *Proteins*, 33:227–239, 1998.

A. Leaver-Fay, B. Kuhlman, and J. Snoeyink. An adaptive dynamic programming algorithm for the side-chain placement problem. In *Pacific Symposium on Biocomputing 10*, pages 16–27. World Scientific Publishing, 2005.

V. Lepar and P. P. Sehnoy. A comparison of lauritzen-spigelhalter, hugin, and shenoy-shafer architectures for computing marginals of probability distributions. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 328–333. Morgan Kaufmann, 1998.

W. A. Lim, A. Hodel, R. T. Sauer, and F. M. Richards. The crystal structure of a mutant protein with altered but improved hydrophobic core packing. *Proc. Natl. Aacd. Sci. U.S.A.*, 91:423–427, 1994.

S. M. Lippow and B. Tidor. Progress in computational protien design. *Curr. Opin. Struct. Biol.*, 18:1–7, 2007.

S. M. Lippow, K. D. Wittrup, and B. Tidor. Computational design of antibody-affinity improvement beyond in vivo maturation. *Nat. Biotechnol.*, 25:1171–1176, 2007.

L. L. Looger and H. W. Hellinga. Generalized dead-end elimination algorithms make large-scale protein side-chain structure prediction tractable: implications for protein design and structural genomics. *J. Mol. Biol.*, 307:429–445, 2001.

A. Lucena. Non delayed relax-and-cut algorithms. *Annals of Operations Research*, 140:375–410, 2005.

A. D. Mackerell, D. Bashford, M. Bellott, R. L. Dunbrack, J. D. Evanseck, M. J. Field, S. Fischer, J. Gao, H. Guo, S. Ha, D. Joseph-Mccarthy, L. Kuchnir, K. Kuczera, F. T. K. Lau, C. Mattos, S. Michnick, T. Ngo, D. T. Nguyen, B. Prodhom, W. E. Reiher, B. Roux, M. Schlenkrich, J. C. Smith, R. Stote, J. Straub, M. Watanabe, J. Wiorkiewicz-Kuczera, D. Yin, and M. Karplus. All-atom empirical potential for molecular modeling and dynamics studies of proteins. *J. Phys. Chem. B*, 102: 3586–3616, 1998.

A. L. Main, T. S. Harvey, M. Baron, J. Boyd, and I. D. Campbell. The 3-dimensional structure of the 10th type-III module of fibronectin — an insight into RGD-mediated interactions. *Cell*, 71(4):671–678, 1992.

S. M. Malakauskas and S. L. Mayo. Design, structure and stability of a hyperthermophilic protein variant. *Nat. Struct. Biol.*, 5:470–475, 1998.

E. Martins and M. Pascoal. A new implementation of yen's ranking loopless paths algorithm. *4OR ̌2013 Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(2):121–134, 2003.

N. Meggiddo and K. J. Supowit. On the complexity of some common geometric location problems. *SIAM Journal of Computing*, 13:182–196, 1984.

Joaquim Mendes, Cláudio M. Soares, and Maria Arménia Carrondo. Improvement of side-chain modeling in proteins with the self-consistent mean field theory method based on an analysis of the factors influencing prediction. *Biopolymers*, 50:111–131, 1999.

A. Nedic and D. P. Bertsekas. Incremental subgradient methods for nondifferentiable optimization. *SIAM Journal on Optimization*, 12(1):109–138, 2001.

G. L. Nemhauser and L. A. Wolsey. Wiley, N. Y., 1988.

C. Pabo. Designing proteins and peptides. *Nature*, 301:200, 1983.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, San Mateo, 1988.

R. W. Peterson, P. L. Dutton, and A. J. Wand. Improved side-chain prediction accuracy using an ab initio potential energy function and a very large rotamer library. *Protein Sci.*, 13:735–751, 2004.

N. A. Pierce, J. A. Spriet, J. Desmet, and S. L. Mayo. Conformational splitting: a more powerful criterion for dead-end elimination. *J. Comput. Chem.*, 21:999–1009, 2000.

J. J. Plecs, P. B. Harbury, P. S. Kim, and T. Albert. Structural test of the parameterized-backbone method for protein design. *J. Mol. Biol.*, 342:289–297, 2004.

J. W. Ponder and F. M. Richards. Tertiary templates for proteins: use of packing criteria in the enumeration of allowed sequences for different structural classes. *J. Mol. Biol.*, 193(4):775–791, 1987.

T. K. Ralphs and M. V. Galati. Decomposition and dynamic cut generation in integer linear programming. *Mathematical Programming*, 106(2):261–285, April 2006.

J. S. Richardson and D. C. Richardson. The de novo design of protein structures. *Trends Biochem. Sci.*, 14:304–309, 1989.

C. S. Ring and F. E. Cohen. Modeling protein structures: construction and their applications. *FASEB J.*, 7:783–790, 1993.

N. Robertson and P. D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In *Graph Theory and Computing*, pages 183–217. Academic Press, 1972.

S. A. Ross, C. A. Sarisky, A. Su, and S. L. Mayo. Designed protein g core variants fold to native-like structures: Sequence selection by orbit tolerates variation in backbone specification. *Protein Sci.*, 10:450–454, 2001.

C. T. Saunders and D. Baker. Recapitulation of protein family divergence using flexible backbone protein design. *J. Mol. Biol.*, 346:631–644, 2005.

H. D. Sherali and G. Choi. Recovery of primal solutions when using subgradient optimization methods to solve lagrangian duals of linear programs. *Operations Research Letters*, 19:105–113, 1996.

J. M. Shifman and S. L. Mayo. Exploring the origins of binding specificity through the computational redesign of calmodulin. *Proc. Natl. Aacd. Sci. U.S.A.*, 100: 13274–13279, 2003.

A. M. Slovic, H. Kono, J. D. Lear, J. G. Saven, and W. F. DeGrado. Computational design of water-soluble analogues of the potassium channel kcsa. *Proc. Natl. Aacd. Sci. U.S.A.*, 101:1828–1833, 2004.

D. Sontag and T. Jaakkola. New outer bounds on the marginal polytope. *Advances in Neural Information Processing Systems (NIPS) 21*, 2007.

S. Subbarayan and H. R. Andersen. Backtracking procedures for hypertree, hyperspread and connected hypertree decomposition of csps. *IJCAI 2007, Hyderabad*, pages 180–185.

R.S. Syed, S.W. Reid, C. Li, J.C. Cheetham, K.H. Aoki, B. Liu, H. Zhan, T.D. Osslund, A.J. Chirino, J. Zhang, J. Finer-Moore, S. Elliott, K. Sitney, B.A. Katz, D.J. Matthews, J.J. Wendoloski, J. Egrie, and R.M. Stroud. Efficiency of signaling through cytokine receptors depends critically on receptor orientation. *Nature*, 395: 511–516, 1998.

R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal of Computing*, 13:566–579, 1984.

C. A. Voigt, D. B. Gordon, and S. L. Mayo. Trading accuracy for speed: a quantitative comparison of search algorithm in protein sequence design. *J. Mol. Biol.*, 299:789–803, 2000.

M. J. Wainwright and M. I. Jordan. Log-determinant relaxation for approximate inference in discrete markov random fields. *IEEE Transactions on Signal Processing*, 54(6):2099–2109, June 2006.

M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. Technical Report 649, UC Berkeley, Dept. of Statistics, 2003.

M. J. Wainwright, T. Jaakkola, and M. I. Jordan. Tree consistency and bounds on the performance of the max-product algorithm and its generalizations. *Stat. Comput.*, 14:143–166, 2004.

M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. Map estimation via agreement on (hyper)trees: Message-passing and linear programming approaches. *IEEE Trans. Inform. Theory*, 51(11):3697–3717, 2005.

S. J. Weiner, P. A. Kollman, D. A. Case, U. C. Singh, C. Ghio, G. Alagona, S. Profeta, Jr., and P. Weiner. A new force field for molecular mechanical simulation of nucleic acids and proteins. *J. Am. Chem. Soc.*, 106(3):765–784, 1984.

S. J. Weiner, P. A. Kollman, D. T. Nguyen, and D. A. Case. An all atom force field for simulations of proteins and nucleic acids. *J. Comput. Chem.*, 7(2):230–252, 1986.

Yair Weiss. Comparing the mean field method and belief propagation for approximate inference in MRFs. In *Advanced Mean Field Methods: Theory and Practice*, pages 229–240. The MIT Press, 2001.

Yair Weiss. Belief propagation and revision in networks with loops. Technical Report AI Memo No. 1616, Massachusetts Institute of Technology, 1997.

L. Wernisch, S. Hery, and S. J. Wodak. Automatic protein design with all atom force field by exact and heuristic optimization. *J. Mol. Biol.*, 301:713–736, 2000.

W. Xie and N. V. Sahinidis. Residue-rotamer-reduction algorithm for the protein side-chain conformation problem. *Bioinformatics*, 22(2):188–194, 2006.

J. Xu. Rapid protein side-chain packing via tree decomposition. In *Proceedings of RECOMB 2005*, pages 423–439. Springer, 2005.

C. Yanover and Y. Weiss. Approximate inference and protein-folding. In *Proceedings of NIPS 2002*, 2002.

C. Yanover, T. Meltzer, and Y. Weiss. Linear programming relaxations and belief propagation. *Journal of Machine Learning Research*, 7:1887–1907, 2006.

J. S. Yedidia, W. T. Freeman, and Y. Weiss. Bethe free energy, kikuchi approximations and belief propagation algorithms. Technical Report TR2001-16, MERL, 2001.

X. Zaho, P. B. Luh, and J. Wang. Surrogate gradient algorithm for lagrangian relaxation. *Journal of Optimization Theory and Applications*, 100(3):699–712, 1999.