An Adaptive Dynamic Programming Algorithm for the Side Chain Placement Problem

A. Leaver-Fay, B. Kuhlman, and J. Snoeyink

Pacific Symposium on Biocomputing 10:16-27(2005)

AN ADAPTIVE DYNAMIC PROGRAMMING ALGORITHM FOR THE SIDE CHAIN PLACEMENT PROBLEM

ANDREW LEAVER-FAY BRIAN KUHLMAN JACK SNOEYINK

UNC-Chapel Hill
Chapel Hill, NC 27514, USA
leaverfa@email.unc.edu
bkuhlman@email.unc.edu
snoeyink@email.unc.edu

Larger rotamer libraries, which provide a fine grained discretization of side chain conformation space by sampling near the canonical rotamers, allow protein designers to find better conformations, but slow down the algorithms that search for them. We present a dynamic programming solution to the side chain placement problem which treats rotamers at high or low resolution only as necessary. Dynamic programming is an exact technique; we turn it into an approximation, but can still analyze the error that can be introduced. We have used our algorithm to redesign the surface residues of ubiquitin's beta sheet.

1. Introduction

Current techniques for protein redesign fit new amino acids onto the rigid backbone scaffold of a known protein. Side chain conformations are almost always modeled by a rotamer library that discretely samples the conformation space. These libraries are obtained from observed side chain conformations in solved structures and/or by quantum calculations with small molecules.[5,15] The energy of a placement depends on the rotamer (or state, S_v) assigned to each residue v, and can be expressed as a sum of internal/background energies for each rotamer and interaction energies between pairs of residues:

$$\sum_v \mathcal{E}_{self}(S_v) + \sum_{v_1<v_2} \mathcal{E}_{pair}(S_{v_1}, S_{v_2}). \quad (1)$$

Minimizing energy expression (1) is the *side chain placement problem*. There are two natural subproblems: the *rotamer relaxation problem*, where the search is only over conformational space, and the *redesign problem* that explicitly involves the additional search over sequence space.

Many computational methods have been applied to side chain placement, including simulated annealing (SA),[2,8,16] dead-end elimination (DEE),[4,14] branch-and-bound,[7] potentials of mean force,[10] genetic algorithms,[3] and integer linear programming.[6] We are not aware of a viable dynamic programming[1] (DP) solution that scales to the size of practical rotamer libraries. DP, unlike all other methods except DEE and branch-and-bound, is guaranteed to find the global optimal state assignment. To make DP viable, we capture all pairs with non-negligible interactions in an *interaction graph*, which guides the dynamic programming as we describe in the next section. This is an extension of our previous work on the hydrogen placement problem.[13] For side chain placement, we make the algorithm adaptively explore the rotamers at either high or low resolution.

2. Methods

2.1. *Interaction Graph Formulation*

A *graph* $G = \{V, E\}$ consists of a set V of *vertices* and $E \subset V \times V$ of *edges.* We say that vertices u and v are *adjacent* if the pair $(u, v) \in E$. A *hypergraph* is a generalization of a graph in which an edge (sometimes called a *hyperedge*) can contain any number of vertices of V. The *degree* of a hyperedge is the number of vertices it is incident upon. In Fig. 1b, we draw a hypergraph with the vertices as points and the hyperedges as curves encircling them.

We can initially create an *interaction graph*, a hypergraph, $G = \{V, E\}$, that captures energy expression (1). Each residue is represented by a vertex, $v \in V$, and each vertex carries a state space $\mathcal{S}(v)$, which is the set of rotamers that can be placed at v.

We initially create a (hyper)edge for each vertex and each pair. We assign each (hyper)edge $e \in E$ a *scoring function* $f_e(S)$ that maps the states chosen for its vertices to their interaction potentials: f_e: $\prod_{v \in e} \mathcal{S}(v) \rightarrow \Re$. Specifically, each vertex, $v \in V$, has a corresponding degree-1 hyperedge, $\{v\} \in E$, with a hyperedge scoring function $f_{\{v\}}(S_v) = \mathcal{E}_{self}(S_v)$. Each pair of vertices, $v_1, v_2 \in V$, has a corresponding degree-2 hyperedge, $\{v_1, v_2\} \in E$, with a hyperedge scoring function $f_{\{v_1,v_2\}}(S_{v_1}, S_{v_2}) = \mathcal{E}_{pair}(S_{v_1}, S_{v_2})$.

We can omit an edge e when f_e is zero under all possible state assignments to the vertices of e. To ignore small contributions, we can also omit edges for which $|f_e|$ is always less than a chosen magnitude threshold, μ. Figure 1a shows an interaction graph for ubiquitin design.

We denote a state assignment to all vertices by S_V, and the induced as-

signment to any subset (particularly hyperedges) e by S_e. Any assignment induces a score for the interaction graph, $\sum_{e \in E} f_e(S_e)$, and the optimal assignment minimizes the score.

2.2. *Dynamic Programming*

A dynamic programming (DP) algorithm can optimize an interaction graph by *eliminating* vertex v from the graph by solving for the optimal state of v for all possible states of its neighbors, then replacing v with a hyperedge containing these neighbors.

Specifically, let E_v be the set of hyperedges that contain v, and $N_v = \bigcup_{e \in E_v} e \setminus \{v\}$ be the neighbors of v. If N_v is not already a hyperedge, create N_v with an initial scoring function $f_{N_v} = 0$. Next, add to f_{N_v} the scoring functions of E_v with the best assignment to v, and eliminate v and edges E_v from the hypergraph. Let $\hat{f}_{e,v=s}$ denote the function whose domain is the states of N_v obtained from f_e by restricting the state of v to be $s \in \mathcal{S}(v)$. Then we can write $f_{N_v} = \min_{s \in \mathcal{S}(v)} \sum_{e \in E_v} \hat{f}_{e,v=s}$. Also record v's optimal state as a function of its neighbors' states for later retrieval.

The scoring function for N_v now represents the simultaneous interaction of the vertices of N_v with the optimal state of v. The minimum score for this reduced interaction graph is the same as the score of the original. Each function is represented in a multi-dimensional table; the table dimensionality is the degree of its corresponding hyperedge. Computing f_{N_v} amounts to filling all cells in the table representing it.

We can reduce a graph to a single vertex by repeated elimination; Figure 1 illustrates the first two vertex eliminations. Once we have the optimal state of this remaining vertex, we can trace back the optimal states for eliminated vertices by reading off their optimal states in the reverse order of their elimination.

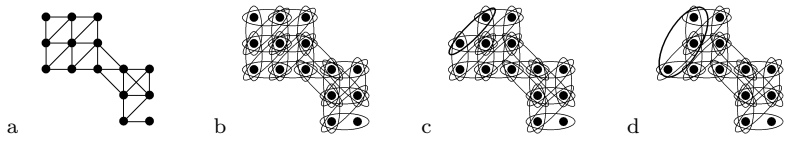

Figure 1. Interaction graph reduction. *The edges in (a) are drawn as curves encircling the vertices they contain in (b). The DP algorithm eliminates the upper-left vertex of (b) and updates the existing degree-2 hyperedge (c). It creates a degree-3 hyperedge as it eliminates the next vertex (d).*

In the remainder of this subsection we give upper bounds on the running time and memory for an interaction graph with n vertices. If we assume

at most s states per vertex, and at most w neighbors at each elimination, then DP runs in $O(ns^{w+1})$ time and uses $O(ns^w)$ space.

The parameter w is known as the *treewidth*[9] of the interaction graph. Our algorithm shows that side chain placement is another instance of an NP-Hard problem with a polynomial time solution for graphs of bounded treewidth.

To define treewidth, we first define the *tree decomposition* of a hypergraph $G = \{V, E\}$ as a tree with T whose vertices $\{X_1, X_2, \ldots, X_m\}$ represent subsets of V that satisfy the following three properties:

(1) The union of sets $\bigcup_{1 \le i \le m} X_i = V$.
(2) Each edge $e \in E$ is in some set: $e \subseteq X_i$ for some $1 \le i \le m$.
(3) Each vertex $v \in V$ occupies a connected part of tree T: for any X_j on a path in T from X_i to X_k, the intersection $X_i \cap X_k \subseteq X_j$.

The treewidth of a tree decomposition is $\max_i |X_i| - 1$. The treewidth of a graph G is the minimum treewidth over all possible tree decompositions of G. Figure 2b illustrates a treewidth-3 tree decomposition of the interaction graph in Fig. 2a.

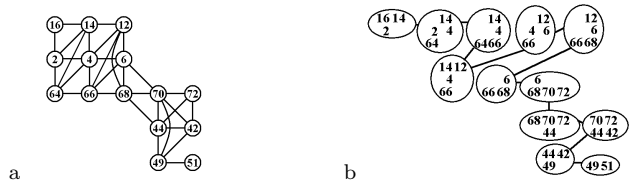

Figure 2. Tree decomposition. *The tree in (b) is a tree-decomposition of the labeled graph in (a). Each tree node contains the vertex label set, X_i.*

To show that bounded-treewidth interaction graphs yield polynomial time optimization, we convert a tree decomposition into a canonical form, which will then specify the elimination order for the vertices of the graph.

Lemma 2.1. *A tree decomposition T with treewidth w of the graph G can be converted into a rooted tree decomposition with treewidth w in which the vertex sets X_i satisfy $|X_i \setminus X_j| = 1$ whenever X_j is the parent of X_i.*

Proof. We give a constructive proof. Begin by choosing an arbitrary node of T as the root. Now, consider any node X_i and parent X_j for which $|X_i \setminus X_j| \neq 1$. If $|X_i \setminus X_j| = 0$, then delete X_i and connect any children of X_i to the parent X_j. Otherwise take one vertex from the set difference, $v \in X_i \setminus X_j$, and create a new node, $Y = (X_i \cap X_j) \cup \{v\}$ between X_i and X_j.

Node X_i now has the desired property. Tree T is still a tree decomposition: the first two properties for tree decompositions hold trivially, and the third property holds because $(X_i \cap X_j) = (X_i \cap Y \cap X_j)$. The treewidth of the new decomposition is still w since $|Y| < |X_i| \leq (w+1)$. Recurse on Y until the desired property holds. □

Now, a depth-first traversal of a rooted, canonical tree decomposition gives an order for vertex elimination: Reaching a node, write down the one vertex in the set difference between it and its parent. At the root, r, write down the vertices of X_r in any order. Because each vertex appears in a connected part of the tree, each vertex is written down once, and the recorded list of vertices provides the elimination order.

Theorem 2.1. *For an interaction graph G on n vertices with at most s states per vertex and a tree decomposition T of treewidth w, we can compute the optimum state assignment in $O(ns^{w+1} + wn)$ time and $O(ns^w + wn)$ space by dynamic programming.*

Proof. The $O(wn)$ terms come from the time and space to construct and use the canonical tree decomposition. To bound the running time of DP, we use induction on the number of nodes eliminated so far. Before eliminating node v, we assert that G is the interaction graph that results from having eliminated the vertices up to vertex v, and that the canonical tree decomposition T that we maintain is a tree decomposition of treewidth w for G. If no vertices before v have been eliminated, then our assertion is trivially true. Vertex v was chosen for elimination because it was contained in a leaf node X_i of T, but not in the parent of X_i. If X_i was the root, then X_i has fewer than $w+1$ nodes, and we may apply brute force optimization in $O(s^{w+1})$ time. Otherwise, let X_j be X_i's parent. By definition of the canonical tree decomposition, $\{v\} = X_i \setminus X_j$. Therefore v is adjacent to at most w vertices, and we can record the best state for v for each assignment to these vertices in $O(s^{w+1})$ time and $O(s^w)$ space. The hyperedge that v's elimination produces is a subset of X_j, so we can delete X_i from T to obtain a tree decomposition of G after the elimination of v. Since each vertex is eliminated once, the theorem is established. □

Although computing the treewidth of a graph is NP-hard, the interaction graphs we have observed are small enough that we can make low-treewidth tree decompositions by hand or by heuristics and feed them as input to our algorithm.

2.3. *The Move Towards Large Rotamer Libraries*

The analysis of the DP algorithm shows that it is principally limited by two parameters: the number of states per node, s, which can be in the tens to the thousands, and the treewidth, w, which is fixed by the interaction graph and where we have only tried problems when it was between three to five. Since the number of vertices n is in the tens for a redesign problem, we seek to reduce the impact of increasing s. How and why does s increase?

Large rotamer libraries are typically cousins of the canonical rotamer libraries,[5,15] obtained by sampling around the canonical rotamers. Specifically, rotamers are usually defined as particular values of χ dihedral angles for side chains of amino acids. A large rotamer library might expand the canonical leucine rotamer represented by χ angles $(60, 180)$ into many samples inside the box $(60 \pm 5, 180 \pm 10)$. The canonical rotamers represent the bottoms of shallow energy wells for side chain conformations; the large rotamer libraries sample these wells. We can therefore organize our rotamer libraries as a set of base-rotamers (boxes) which contain many sub-rotamers (points): we denote the sub-rotamers of a base-rotamer b by subscripts $b_1, b_2, \ldots$

Protein designers have found that large rotamer libraries produce better designs. The improvement can be attributed to the high penalty assigned to colliding atoms. The Lennard-Jones "6-12" potential suggests a larger rotamer library: the $\frac{1}{d^{12}}$ repulsive term is sensitive to small changes when d is small, and slight flexes of the χ dihedrals resolve some high penalty collisions. The remaining terms of the energy functions are less sensitive: electrostatic interaction, for instance, is a function of $\frac{1}{d}$. Thus, for rotamers that are not near each other, we hope to represent many sub-rotamer interactions by a single base-rotamer interaction. We formalize this idea by defining the concept of stiff rotamer interactions.

2.4. *Stiff Interactions*

An assignment of base-rotamers, a, b, to the adjacent vertices u and v is *stiff in* $\{u, v\} \in E$ if any sub-rotamers a_i, a_j and b_k, b_l satisfy

$$|f_{\{u,v\}}(a_i, b_k) - f_{\{u,v\}}(a_j, b_l)| > \epsilon.$$

If a base-rotamer interaction fails to exceed the stiffness threshold, then we will approximate the energy between pairs of sub-rotamers by the energy between the canonical rotamers and ϵ will bound the error in this approximation.

Our analogy is with stiff differential equations: the stiff interactions are those where the energy is rapidly changing over a collection of sub-rotamers. As adaptive schemes for numerical integration increase their temporal resolution when their input ODEs become stiff, and decrease their resolution when the ODEs behave smoothly, our algorithm increases its spatial resolution when it encounters stiff base-rotamer combinations, examining all possible sub-rotamer combinations, and decreases its resolution for non-stiff base-rotamer combinations.

2.5. *Adaptive Dynamic Programming*

In adaptive dynamic programming, each vertex carries two levels of state spaces. At the top level is the base-state state space, $\mathcal{L}(v)$, and each base-state $b \in \mathcal{L}(v)$ carries a sub-state state space, $\mathcal{H}(b)$. The hyperedge scoring functions are defined as a mapping $f_e : \prod_{v \in e} \mathcal{L}(v) \rightarrow (\prod_{b \in B} \mathcal{H}(b) \rightarrow \Re)$ where B, a base state assignment to the vertices in e, is the input argument to the outer function. For each edge e and each base-state assignment B, we maintain a *stiffness descriptor*, C_e^B which is a family of subsets of B. Let $C_{e,k}^B \subseteq B$ represent the k^{th} element of C_e^B where $1 \leq k \leq |C_e^B|$. Alongside the stiffness descriptor, we define the set of all stiff base-states in an assignment, $U_e^B = \bigcup_{k=1}^{|C_e^B|} C_{e,k}^B$.

If e is an input hyperedge $\{u, v\}$ under the base-state assignment $\{a, b\}$, the stiffness descriptor is either $C_{\{u,v\}}^{\{a,b\}} = \{\{a, b\}\}$ if a and b's interaction is stiff or $\{\{\}\}$ otherwise. A hyperedge created over the course of the reduction corresponds to an eliminated vertex; the stiffness descriptors for such a hyperedge correspond to the stiffnesses that existed with the optimal states of the eliminated vertex.

The elimination of vertex v with incident edges E_v leaves behind a hyperedge N_v containing v's former neighbors. We will describe the stiffness descriptor for a single base-state assignment, B, to the vertices of N_v. Define the stiffly-interacting base-states of a single base-state of v, $b \in \mathcal{L}(v)$ to be $T_{E_v}^{(B,b)} = \cup_{e \in E_v} U_e^{(B,b)_e} - b$ where $(B, b)_e$ represents the base-states assigned to the vertices e contains. Let P^B represent the power set of B, where if $|N_v| = d$ then $|P^B| = 2^d$, and let P_i^B be the i^{th} member of P_i^B. Let $\mathcal{L}_{P_i^B}(v)$ be $\{b \mid T_{E_v}^{(B,b)} = P_i^B\}$, that is, the set of all base-states of v which have the same stiffly-interacting base-states of P_i^B.

Let $\hat{f}_e((B, b)_e)_{v=s}$ denote the function whose domain is the sub-states of N_v under the base-state assignment B obtained from $f_e((B, b)_e)$ by restricting v to sub-state s. Then for each i with a non-empty $\mathcal{L}_{P_i^B}(v)$, compute

the function $f_{P_i^B} : \prod_{b \in P_i^B} \mathcal{H}(b) \rightarrow \Re$ so that

$$f_{P_i^B} = \min_{b \in \mathcal{L}_{P_i^B}(v)} \min_{s \in \mathcal{H}(b)} \sum_{e \in E_v} \hat{f}_e((B,b)_e)_{v=s}$$

We represent the range of $f_{P_i^B}$ by $[\text{best}_{P_i^B} \ldots \text{worst}_{P_i^B}]$. We define the *best-worst* score as $\min_i \text{worst}_{P_i^B}$ and define the set of competing stiffnesses as $I = \{i \mid \text{best}_{P_i^B} < \textit{best-worst}\}$.

Now, there is a natural partial order based on the subset property for power sets. We use this partial order to define a *maximal* family of sets to be any where no element in the family is a subset of any other. Given a family of sets, P we define maximal(P) to be the function which returns the maximal family produced by throwing out any set if it is a subset of any other. The stiffness descriptor $C_{N_v}^B$ will be assigned maximal($\{P_i^B \mid i \in I\}$). Let the set I_k, corresponding to $C_{N_v,k}^B$, represent $\{i \mid P_i^B \subseteq C_{N_v,k}^B\}$. Then we define a set of functions, $f_{C_{N_v,k}^B} = \min_{i \in I_k} f_{P_i^B}$. Then the hyperedge scoring function with the state assignment B can be defined as $f_{N_v}(B) = \min_k f_{C_{N_v,k}^B}$.

Where we represented each hyperedge scoring function before as a multi-dimensional table, we now represent each function as multi-resolution multi-dimensional table: a table of tables. The top level table has an entry for each base-state assignment to the vertices the edge contains. In each entry resides a set of tables holding the $f_{C_{N_v,k}^B}$ functions. Each table requires $\prod_{b \in C_{N_v,k}^B} |\mathcal{H}(b)|$ space. The smaller tables reduce the memory required by the adaptive algorithm compared to standard dynamic programming.

2.6. *Irresolvable Collisions*

With a final parameter, τ, we define a pair of base-rotamers b and c to be in an irresolvable collision if

$$\neg\exists_{i,j} \mathcal{E}_{pair}(b_i, c_j) < \tau$$

While some pairs of base-rotamers can resolve their collisions by slight dihedral flexes, others cannot. As long as we have hope that a collision-free placement of side chains exists, we need not examine the colliding pairs.

Within the DP framework, this means we may avoid calculating the best state of a vertex for any combination of neighbors's base-states that put them in an irresolvable collision. Since the $\frac{1}{d^{12}}$ collision term is fluctuating wildly, irresolvable collisions will meet our stiff interaction threshold, ϵ, and we would wastefully treat them at high resolution if we did not ignore them.

2.7. *Error Analysis*

Theorem 2.2. *The score induced by the adaptive algorithm's state assignment S_V is within $2\epsilon|E_2|$ of the global optimum score where $|E_2|$ is the number of degree-2 hyperedges in the input interaction graph.*

Proof. Let σ_{S_V} be the energy of the adaptive algorithm's state assignment and σ_{opt} be the global minimum energy. Now, in selecting a sub-state of some base-state and representing it's interaction energy with another base-state non-stiffly, the algorithm can misrepresent the energy by at most ϵ. On one hand, this may decrease the apparent score of the state assignment S_V while on the other increasing the apparent score of the global optimal state. Over the course of the optimization, the algorithm may misrepresent at most $|E_2|$ non-stiff interactions, so the apparent score of S_V may decrease to $\sigma_{S_V} - \epsilon|E_2|$ and the apparent score of the optimal state may increase to $\sigma_{opt} + \epsilon|E_2|$. Because the algorithm chose S_V, we know

$$\sigma_{S_V} - \epsilon|E_2| \leq \sigma_{opt} + \epsilon|E_2|$$

and thus $\sigma_{S_V} - \sigma_{opt} \leq 2\epsilon|E_2|$. □

3. Results

We tested our algorithm at the rotamer relaxation task and the redesign task. For both tasks, we selected 15 surface residues from ubiquitin's β-sheet, pictured in Fig. 3a. We excluded the following amino acids to keep the treewidth of our interaction graphs low: arginine, lysine, and methionine.

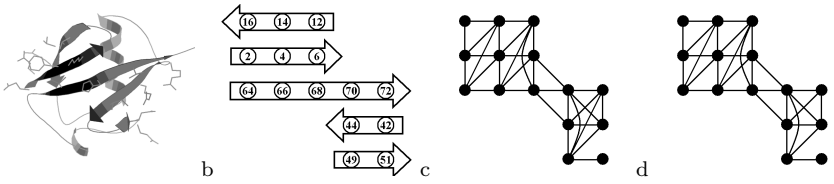

Figure 3. Ubiquitin's β-sheet. *The β-sheet in (a) is flattened in (b) with it's 15 surface residues shown. We observed the treewidth-4 interaction graph in (c) by including edges between residues if any pair of rotamers ever interacted with an energy magnitude at least $\mu = 0.2$ kcal/mol. We artificially created the treewidth-3 interaction graph in* (d) *by dropping a single edge.*

For the rotamer relaxation task, we first created 100 sequences for the ubiquitin backbone, asking the design module of the Rosetta molecular

modeling program[11] to stochastically redesign these 15 surface residues. We then evaluated Rosetta's experimentally validated energy function[12] between all pairs of sub-rotamers, and included hyperedges that met our interaction magnitude threshold $\mu = 0.2$ kcal/mol. This produced a treewidth-4 interaction graph, shown in Fig. 3c. We set our irresolvable collision cutoff to $\tau = 1$ kcal/mol. We compared the standard DP algorithm against the adaptive algorithm with ϵ values of 0, 10^{-4}, 10^{-3}, 10^{-2}, 0.1, and 1.0. Against DP, we compared the time a single Rosetta SA design required and scores it produced.

In the relaxation problem, the average residue had 32 total rotamers, breaking down into 3 base-states and 10 sub-states per base-state. The median state space size was $\sim 10^{18}$. We measured performance on a dual 2 GHz AMD Athlon with 2 GB RAM. In Fig. 4 we plot the relative running time of the adaptive and standard DP algorithms against the actual error observed. In table 1, we present the actual running times. Except for three instances, SA produced the optimal answer when run for as long as standard DP.

Table 1. Average running time comparison, in milliseconds, at the rotamer relaxation task.

Run Time	DP	$\epsilon = 0$	10^{-4}	10^{-3}	10^{-2}	0.1	1.0	SA
Mean	206.2	63.7	62.9	63.0	61.2	17.5	6.4	65.1
Median	117.3	53.7	53.2	53.7	50.7	11.3	4.8	65.0
Std Dev	399.3	49.1	48.6	48.9	48.8	38.2	7.6	6.5

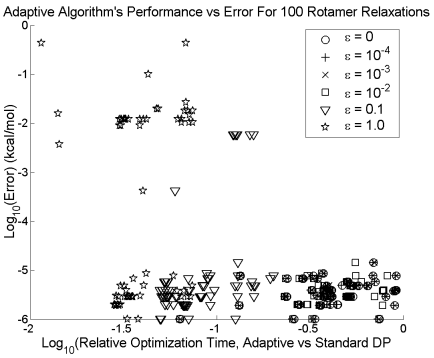

Figure 4. Rotamer relaxation task. *Increasing ϵ to as high as 1.0 kcal/mol gives a theoretical error bound of ± 64 kcal/mol but actually preserves accuracy and greatly decreases running time.*

For the redesign task, we artificially imposed a treewidth-3 interaction graph on the problem, pictured in figure 3d. This interaction graph differs by a single edge from the graph in 3c. The absence of this edge decreases the quality of the design. We none the less include the task as it pushes the DP algorithm's limits.

Each residue in the design problem averaged 680 total rotamers. This broke down into about 57 base-rotamers per residue and 12 sub-rotamers per base-rotamer. The size of the state space was $\sim 10^{42}$. We measured the performance of both the standard and the adaptive DP algorithms on a dual 900 MHz 64-bit Itanium-2 with 10 GB of RAM. We compared against a single SA run on the 2 GHz Athlon. In table 2 we present the results.

Table 2. Redesign task performance comparison.

	DP	$\epsilon = 0.1$	$\epsilon = 1.0$	SA
Run Time	15.99 hrs.	5.07 hrs.	1.52 hrs.	3.42 seconds
Memory Usage	3.7 GB	3.4 GB	1.5 GB	0.2 GB
Score (kcal / mol)	-42.5893	-42.5893	-42.5579	-42.5692
Error (kcal / mol)		0.0000	0.0314	0.0201

4. Discussion

We have presented a novel application of our DP algorithm, and an improvement upon it that begins to make it competitive. DP offers an alternative to DEE and branch-and-bound for finding the global optimal solution in the side chain placement problem. The other algorithms are designed to solve a very difficult problem; the generic interaction graph implied by energy expression (1) is fully connected. Distant amino acids, however, have interaction energies of zero, so the interaction graphs we encounter are sparsely connected. Actual instances of the side chain placement problem may not require algorithms as generic as DEE and branch-and-bound.

We were surprised to observe the small error for high values of ϵ. None of our experiments with $\epsilon > 10^{-2}$ produced an error near the theoretical bound we proved in Sec. 2.7. When we set ϵ as high as 1 kcal/mol, we limit our high resolution focus to only those sometimes-colliding pairs of base-rotamers. It is possible that in general the interactions in the global optimal solution induces are either very stiff or almost totally insensitive to small change; rotamers will either pack tightly or are far apart. It is also possible that the non-canonical rotamers' internal strain prevents their selection except in the instances of collision resolution. By representing base-state interactions with the canonical rotamer interaction energies, we would introduce error only for the non-stiff interactions induced alongside

a resolved collision—the resolved collision itself would be modeled stiffly.

5. Future Work

We have allowed our adaptive algorithm only two spatial resolutions: high or low. We want to heirarchically group rotamers with varying resolutions dictated by the sub-rotamer energy range. This would let us make tighter theoretical bounds while hopefully preserving our performance gains. Moreover, we want to treat some states at even lower resolution. If two base-rotamers for a residue reached away from the vertex v being eliminated so that their interaction energies with the rotamers of v were the same, we could treat these two base-rotamers as one and reduce the redundant computations.

We would also like to incorporate partial dynamic programming into a simulated annealing algorithm. We can decrease the problem complexity if we allow DP to eliminate all degree-1 and -2 vertices. This fixes the eliminated vertices in their optimal states. We expect this adaptation will produce better designs.

Acknowledgements

This research was partially funded by NSF grant 0076984.

References

1. R. Bellman. *Dynamic Programming*. Princeton Univ. Press, 1957.
2. B. I. Dahiyat and S. L. Mayo. *Science*, **278** 82, (1997).
3. J. R. Desjarlais and T. M. Handel. *Prot. Sci.*, **4** 2006, (1995).
4. J. Desmet, M. D. Maeyer, B. Hazes, and I. Lasters. *Nature*, **356** 539, (1992).
5. R. L. Dunbrack. Jr. *Curr. Opin. Struct. Biol.*, **12** 431, (2002).
6. O. Eriksson, Y. Zhou, and A. Elofsson. In *WABI'01*, 128, (2001).
7. D. B. Gordon and S. L. Mayo. *Structure* **7** 1089, (1999).
8. L. Holm and C. Sander. *Proteins*, **14**(2):213, (1992).
9. T. Kloks. *Treewidth: computations and approximations*. Springer, (1994).
10. P. Koehl and M. Delarue. *J Mol Biol*, **239**(2) 249, (1994).
11. B. Kuhlman, and D. Baker. *Proc Natl Acad Sci USA*, **97** 10383, (2000).
12. B. Kuhlman, G. Dantas, G. C. Ireton, G. Varani, B. L. Stoddard, and D. Baker. *Science*, **302** 1364, (2003).
13. A. Leaver-Fay, Y. Liu, and J. Snoeyink. In *ALENEX'04*, (2004).
14. L. L. Looger and H. W. Hellinga. *J Mol Biol*, **307**(1) 429, (2001).
15. S. C. Lovell, J. M. Word, J. S. Richardson, and D. C. Richardson. *Proteins*, **40** 389, (2000).
16. J. G. Saven and P. G. Wolynes. *J. Phys. Chem. B*, **101** 8375, (1997).