Faster Placement of Hydrogens in Protein Structures by Dynamic Programming

ANDREW LEAVER-FAY, YUANXIN LIU, JACK SNOEYINK, and XUEYI WANG
Computer Science, UNC Chapel Hill

M. Word and coauthors from the Richardsons' 3D Protein Structure laboratory at Duke University propose *dot scores* to measure interatomic interactions in molecular structures. Their program REDUCE uses these scores in a brute-force search to solve instances of the *NP*-hard problem of finding the optimal placement of hydrogen atoms in molecular structures determined by X-ray crystallography. We capture the central combinatorial optimization in the hydrogen placement problem with an abstraction that we call an interaction (hyper)graph. REDUCE's dot-based scoring function cannot be decomposed into the sum of pair interactions, but because the function is short ranged we are able to decompose it into the sum of single, pair, triple, and quadruple interactions that we represent by graph hyperedges. Almost every interaction graph we have observed has had a small treewidth. This fact allows us to replace the brute-force search by dynamic programming, giving speedups of nearly ten orders of magnitude. This dynamic programming has been incorporated into REDUCE and is available for download.

Categories and Subject Descriptors: F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Geometrical problems and computations*

General Terms: Algorithms

Additional Key Words and Phrases: Dynamic programming, treewidth, hard-sphere model, hydrogen bonds, hydrogen placement, protein structure

ACM Reference Format:

Leaver-Fay, A., Liu, Y., Snoeyink, J., and Wang, X. 2007. Faster placement of hydrogens in protein structures by dynamic programming. ACM J. Exp. Algor. 12, Article 2.5 (2007), 16 pages DOI 10.1145/1227161.1227167 http://doi.acm.org 10.1145/1227161.1227167

1. INTRODUCTION

Molecular biologists frequently analyze atom contacts within a protein molecule to accurately determine and evaluate protein structure. Michael Word and others from David and Jane Richardsons' 3D Protein Structure laboratory [Lovell

Authors' address: Andrew Leaver-Fay, Yuanxin Liu, Jack Snoeyink, and Xueyi Wang, Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, North Carolina 27514; email: plato@es-une.edu.

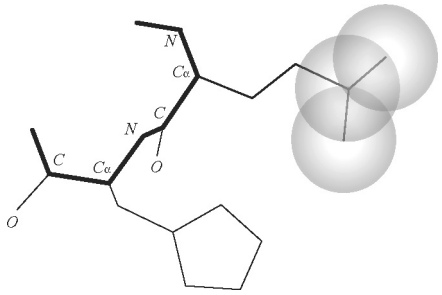


Fig. 1. Protein backbone and side chain bonds with three van der Waals spheres.

et al. 2003; Richardson and Richardson 2001; Word et al. 2000; Word et al. 1999a, 1999b] have developed a set of open-source programs and web tools for atom contact analysis [`http://kinemage.biochem.duke.edu`]. A key program, named REDUCE, adds hydrogens to a protein databank (PDB) molecular structure based on atom contacts [Word et al. 1999b]. It finds the placement of hydrogen atoms that maximizes a scoring function.

REDUCE uses a brute-force search, which is reasonably fast since most inputs require the enumeration of only a small number of hydrogen placements. Unfortunately, there are a number of bad input cases that REDUCE cannot handle in an acceptable amount of time. In this paper, we present a graphical formulation of their optimization problem and a dynamic programming solution. We have incorporated dynamic programming into REDUCE by replacing their scoring and search subroutines. We compared the modified version of REDUCE with the original on these bad input cases. Our program produces identical outputs and is faster by orders of magnitude.

2. REPRESENTATIONS OF PROTEIN STRUCTURE

Biologists speaking of proteins will often say that a protein's sequence determines its structure, which determines function [Branden and Tooze 1999]. The sequence of a protein is the chain of the 20 different amino acids that form the protein; Figure 1 shows two amino acids in a portion of a protein. Each amino acid contains $-N-C_\alpha-C-$ atoms, which form the backbone of the protein. The 20 naturally occurring amino acids differ in their side chains, which are chemical groups bound to the C_α atom. The combination of the side chain and backbone chemical structure determines a protein's physical structure—its fold.

A simple model of atomic structure is the hard-sphere molecular model, in which each atom is modeled as a three-dimensional (3D) sphere with radius equal to the atom's van der Waals radius. Spheres may overlap only for atoms that share a chemical or hydrogen bond. For proteins, this simple geometric model not only eases visualization, but also captures poor interactions between atoms as collisions between spheres.

The software tools developed by the Richardsons' 3D Protein Structure laboratory take protein coordinates as input and analyze the interatom interactions using a hard sphere model. A program named KINEMAGE visualizes a protein model and shows colored dots and spikes for atom contacts between nonbonded

atoms. The dots and spikes are produced by a program named PROBE. A first step for either of these analyses is to run a program named REDUCE, the topic of this paper, which takes a PDB file and adds hydrogens to it. This needs some explanation.

The protein data bank (PDB) is a public repository at `http://www.rcsb.org/pdb` for all experimentally determined protein structures. A PDB file contains the Euclidean coordinates of the protein atoms (in angstroms). The most popular method for structure determination, X-ray crystallography, is based on observing electron density. Crystallographers thread a protein chain through the observed density to create a structural model. When the density predicted from the threaded model matches the observed density, that threading is said to "fit" the density. Density is easily observed for the heavy atoms of proteins—carbon, nitrogen, and oxygen. Electron density is rarely resolved for hydrogen atoms. Thus, hydrogen atoms, are invisible in a standard X-ray diffraction experiment and crystallographers do not include them in the structural models they deposit in the PDB.

This is unfortunate since hydrogens represent about one-half the atoms of a protein-making hydrophobic-packing analysis difficult (or dubious) in their absence. Moreover, some hydrogens form hydrogen bonds, which play a crucial, though controversial, role in protein stability. For example, α-helices and β-sheets are common structural elements characterized by the presence of intra-molecular backbone hydrogen bonds; that they are so common implies they are stable. Analysis of the suitability contribution made by hydrogen bonds is frustrated by the absence of hydrogens in protein structures.

The hydrogen atoms that can participate in hydrogen bonds are those chemically bound to a polar heavy atom, such as oxygen or nitrogen. A hydrogen bond is formed as an overlap between such a hydrogen atom and another polar heavy atom to which the hydrogen is not chemically bound. This second heavy atom is called the hydrogen-bond *acceptor*. The term "hydrogen-bond *donor*" refers sometimes to the hydrogen atom and sometimes to the heavy atom to which it is bound. In Appendix A, we refer to the hydrogen when we use the term hydrogen-bond donor.

REDUCE takes a protein model (the coordinates of all nonhydrogen atoms) from a PDB file and finds the best (defined as maximizing a scoring function) placement of hydrogens onto the model. The output of REDUCE can then be used by PROBE and KINEMAGE to analyze the interaction of all of the protein's atoms [Richardson and Richardson 2001; Word et al. 2000]. The broad goal of REDUCE, when searching for the best hydrogen placement, is to minimize overlap between nonbonded atoms and to maximize the amount of hydrogen bonding.

What choices does REDUCE have to add hydrogens? The positions of most hydrogen atoms are fixed once the coordinates of the bound heavy atoms are known. REDUCE computes these positions with simple vector geometry. However, hydrogens in -OH, -SH and -NH_3^+ groups have rotational freedom (i.e., there is a circle in space on which the hydrogen atom's center lies). REDUCE infers possible positions for these hydrogen atoms from the presence of nearby hydrogen-bond acceptors or obstacles.

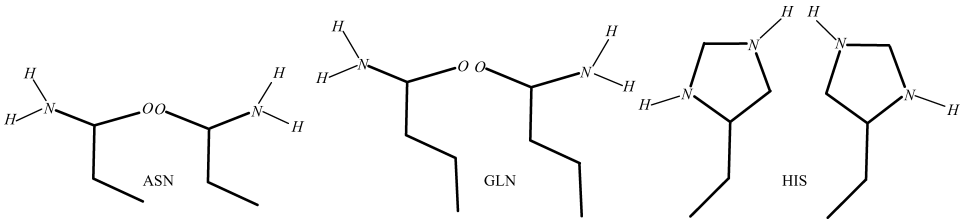


Fig. 2. Flip state pairs for three amino acids.

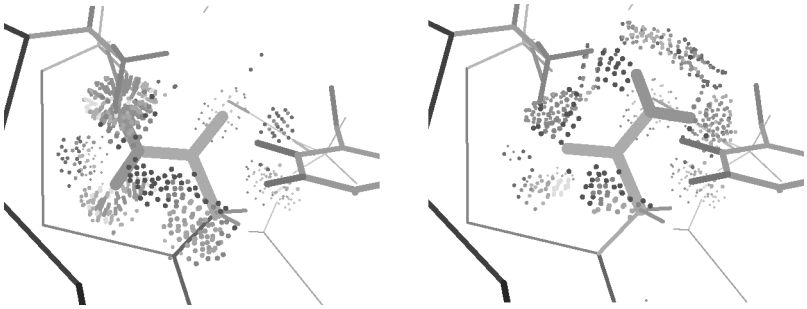


Fig. 3. Asn induces different atom contacts in two "flip" positions. The spikes (red) (*left*) indicate a bad van der Waals overlap; the dot pillows (green) (*right*) indicate favorable hydrogen bonds. (Images from KINEMAGE.)

REDUCE can also resolve another structural ambiguity that arises in X-ray crystallography. Certain asymmetric side chains—those of Asn, Gln, and His, in particular [Branden and Tooze 1999]—appear symmetric within the X-ray crystallographic data, because the observed electron densities for carbon, oxygen, and nitrogen atoms are similar. As Figure 2 illustrates, when the amide groups of the Asn and Gln side chains are *flipped*, the nitrogen and oxygen atoms swap positions, and when the ring of a histidine is flipped, two carbon and two nitrogen atoms swap positions. If a configuration of one of these side chains, having been threaded through the electron density by the crystallographer, fit that density well, then the flipped configuration would fit that density equally well. A crystallographer threading these side chains through the electron density without concern for hydrogen placement will often make an arbitrary choice between the two viable flip positions. REDUCE resolves the flip states for these ambiguous side chains by looking at their interactions with the surrounding atoms.

On the Asn and Gln terminals, the nitrogen-bound hydrogen atoms are able to act as hydrogen-bond donors, and the oxygen atom can act as a hydrogen-bond acceptor. If one flip state produces good hydrogen bonding on both sides of the amide, then the other flip state matches two hydrogen-bond acceptors and pushes two hydrogen-bond donors into an impossible collision (Figure 3). By explicitly modeling the hydrogen atoms for these side chains, REDUCE is able to resolve structural ambiguity for heavy atoms as well.

REDUCE searches for the best hydrogen placement by considering the assignment of states to a collection of movers. A *mover* is a group of atoms that

can be rotated or flipped together, changing their coordinates. Each (discrete) choice of rotation angle for a rotatable atom or flip state for an ambiguous side chain defines the state for a mover. The best hydrogen placement is the configuration of movers that maximizes a scoring function that considers nonbonded overlap and hydrogen bonding. Word et al. present their scoring function as

$$\text{Score} = \sum 4\ \mathit{Vol}(\text{ Hbond Overlap }) - 10\ \mathit{Vol}(\text{ Non-Hbond Overlap})$$

The actual function is a discrete approximation of these volumes. This scoring function is not pairwise decomposable, unlike many of the scoring functions typical in computational chemistry. To accommodate the nonpairwise decomposability, REDUCE's authors chose to optimize using brute force instead of turning to other exact optimization techniques, such as dead-end elimination [Desmet et al. 1992]. We decompose the energy function as much as we can and find that when we are done, the majority of units we have decomposed the function into are singles, pairs, and triples. With the decomposition we create an *interaction graph* holding an abstraction of the original scoring function and perform dynamic programming on the interaction graph.

3. DOT SCORES

Word et al. [1999a, 1999b] define a variation on contact-dot analysis originally developed 15 years earlier [Richardson and Richardson 1987], and describe its use in scoring atomic packing. Initially, between 200 and 600 dots are placed on the van der Waals surface of each atom, A, at roughly uniform density and spacing. Dots that fall inside an atom chemically bound to A are discarded. Each remaining dot d on A finds the closest van der Waals sphere B of an atom not bonded to A. This distance is considered negative if d is inside the van der Waals sphere B. Dot d is assigned a favorable *hydrogen-bond score* if it is inside B (but not too deeply) and the atoms A and B are a hydrogen-bond donor and acceptor. The hydrogen-bond score is weighted by dot d's penetration depth inside B. Dot d is assigned an unfavorable *overlap penalty* if d lies inside B and either A/B is not a hydrogen-bond donor/acceptor pair, or d is too deeply inside B. The overlap penalty score is also weighted by d's penetration depth. (In PROBE, but not in REDUCE, dot d is assigned a favorable *contact score* if it is outside of B, but within a small distance of 0.5 Å.) (Although this dot approximation does not converge to the overlap volume, the biologists are satisfied with it. We analyze the accuracy of the scoring function in more detail in Appendix A).

We will now decompose this scoring function. Let the mover set V of a protein have for each mover $v \in V$ an associated set of states, $\mathcal{S}(v)$. Then

- For a set of movers V', let $\mathcal{S}(V') = \prod_{v \in V'} \mathcal{S}(v)$ denote the state space of the movers in V', and let $S_{V'} \in \mathcal{S}(V')$ denote a particular state vector in the state space of V'.
- Given a set of movers $V' \subseteq V$ and their state vector $S_{V'}$, let $S_{V'' \subseteq V'}$ denote the partial state vector for a subset of V', V''.
- Let $v(s)$ denote the assignment to mover v of state $s \in \mathcal{S}(v)$. Similarly, let $v(S_{v \in V'})$ denote the assignment to mover v of the appropriate state held in the state vector $S_{V'}$, $v \in V' \subseteq V$.

A particular state assignment $v(s)$, fixes the atoms of v in space. Denote the dots placed on the surface of the atoms of $v(s)$ by dots$(v(s))$ and dots$(v) = \cup_{s \in \mathcal{S}(v)}$dots$(v(s))$. The scores of the dots in dots$(v(s))$ depend on the position of the atoms around them, which means they depend on the states of the movers around them. Consider a second mover $v' \neq v$. Dot $d \in$ dots$(v(s))$ is *inside* $v'(s')$ when v' is assigned state $s' \in \mathcal{S}(v')$ if one of the atoms of $v'(s')$ contains d. More generally, d is inside v' if $\exists_{s' \in \mathcal{S}(v')} d$ inside $v'(s')$. We describe the set of movers that influence a dot d as $V_d = \{v : d \in$ dots(v) or d inside $v\}$. The dot score for a dot, *score*(d), is a function of the states of V_d.

By grouping dots together that are influenced by the same movers, we decompose the scoring function. Denote the set of dots that associate with a set of movers $e \subseteq V$, as $D(e) = \{d : V_d = e\}$. The scores for the dots in D_e are determined by the states of e and are independent of the states assigned to all other movers. Define the set of dots in D_e on the surface of the movers in e given some state assignment $S_e \in \mathcal{S}(e)$ to those movers as $D(e \mid S_e) = \{d : d \in D(e)$ and $\exists_{v \in e} d \in$ dots$(v(S_{v \in e}))\}$.

We are ready to define a scoring function for e. Specifically, we define a mapping $f_e \colon \prod_{v \in e} \mathcal{S}(v) \rightarrow \Re$, where

$$f_e(S_e) = \sum_{d \in D(e \mid S_e)} score(d)$$

We define a family of subsets of V, $E = \{e : D(e)$ nonempty$\}$. We observe the protein's score for a particular assignment of states S_V is

$$\sum_{v \in V} \left(\sum_{d \in \text{dots}(v(S_{v \subseteq V}))} score(d) \right) = \sum_{e \in E} f_e(S_{e \subseteq V})$$

This decomposition produces a useful abstraction of the scoring function, an *interaction graph*, $G = \{V, E, \mathcal{S}(V), F\}$. V, and $\mathcal{S}(V)$ are as defined above. The family of subsets of V, E, defines a set of *hyperedges*—edges incident on an arbitrary number of vertices. We say that the *degree* of a hyperedge e is the number of vertices it is incident upon, $|e|$. F is a set of functions so that for each hyperedge $e \in E$ we have $f_e \in F$. We will call these functions *hyperedge scores*.

4. DYNAMIC PROGRAMMING

With the decomposition of the total score into the sum of hyperedge scores, we are ready to formally define the problem.

Problem 4.1. Given an interaction graph, $G = \{V, E, \mathcal{S}(V), F\}$, find the assignment of states to all vertices such that the sum of the hyperedge scores is maximized.

The original REDUCE algorithm found the connected components of the interaction graph and optimized them individually by brute force. We also optimize the connected components individually and will, henceforth, assume that any interaction graph is connected.

We now give a dynamic programming solution to the problem. In this solution, the interaction graph both guides the dynamic programming process and stores the subproblem solutions.

Start with the original interaction graph. At each step, find a vertex x. Let E_x denote the set of hyperedges incident upon x and V_x denote the neighboring vertices of x. Update the graph by *eliminating* vertex x:

1. If V_x is not already a hyperedge, insert V_x into E' and let f_{V_x} be zero for all $\mathcal{S}(V_x)$.
2. For each state $S_{V_x} \in \mathcal{S}(V_x)$, compute the new score $f'_{V_x}(S_{V_x}) =$
$$f_{V_x}(S_{V_x}) + \max_{S_x \in \mathcal{S}(x)} \left(\sum_{e \in E_x} f_e(S_x \times S_{(e-x) \subseteq V_x}) \right)$$
3. Delete the vertex x.

Vertex elimination terminates when only a single vertex remains—and, with it, a single degree-1 hyperedge. In a final backtracking step, reconstruct the optimal state assignment for the entire graph.

Each dynamic programming step reduces the original problem by eliminating x and (re)defining the scoring function for hyperedge V_x, which may need to be created. To show that the new problem instance is equivalent to the original one, we demonstrate an equality. For notational convenience, when the states of vertices in V are fixed, let the resulting set of scores F be partitioned into two: F_x, the scores for hyperedges containing x, and F_{V-x}, the rest of the scores. Then,

$$\begin{aligned} \max_{S_v \in \mathcal{S}(V)} \sum_{i \in F} i &= \max_{S_{V-x} \in \mathcal{S}(V-x)} \left(\max_{S_x \in \mathcal{S}(x)} \left(\sum_{i \in F_{V-x}} i + \sum_{i \in F_x} i \right) \right) \\ &= \max_{S_{V-x} \in \mathcal{S}(V-x)} \left(\sum_{i \in F_{V-x}} i + \max_{S_x \in \mathcal{S}(x)} \sum_{i \in F_x} i \right) \end{aligned}$$

The left-hand side of the equality is the optimal solution to the original problem. We can easily rewrite the right-hand side as the optimal solution to the reduced problem. Given a set of states S_{V-x}, let j denote the score for the hyperedge V_x, if V_x exists and 0 otherwise. The optimal score for the original hypergraph, which is the right hand side above, is then equal to

$$\max_{S_{V-x} \in \mathcal{S}(V-x)} \left\{ \left(\sum_{i \in F_{V-x}} i - j \right) + \left(j + \max_{S_x \in \mathcal{S}(x)} \sum_{i \in F_x} i \right) \right\}$$

This is the optimal score for the modified hypergraph, since the first of the two parenthesized terms is the sum of the unmodified hyperedge scores, and the second is, by definition, the new hyperedge score for V_x.

The running time and memory requirements of this dynamic programming algorithm can be analyzed as follows. When eliminating a vertex x, dynamic programming creates a hyperedge V_x if the hyperedge V_x does not already exist in the graph. Let the *degree* of x be $|V_x|$, the number of neighbors of x

(note that this may be different than the number of incident hyperedges). The created hyperedge, V_x, needs a table of scores whose size is the product of the number of states of the neighbors, $\prod_{v \in V_x} |\mathcal{S}(v)|$, and computing this table takes $\Theta(\prod_{v \in V_x \cup \{x\}} |\mathcal{S}(v)|)$ time. Since both time and space are exponential in the degree of x, it is fastest and cheapest to eliminate vertices with low degree.

The class of graphs that can be reduced to a single vertex by eliminating vertices of degree at most k has been characterized: they are the *partial k-trees*. A *k-tree* is defined recursively: a $(k+1)$-clique is a k-tree, and any graph formed from a k-tree T by connecting a new vertex to each vertex of a k-clique of T is also a k-tree. A partial k-tree is a k-tree from which any number of edges have been removed. The treewidth of a graph, G, is the smallest k such that G is a partial k-tree. The graphs we know as trees are the 1-trees, and forests are partial 1-trees. Computing the treewidth of a given graph is NP-Hard.

For partial 3-trees, there exists an elimination order in which vertices have degree, at most, three. In fact, Arnborg and Proskurowski [1986] gave a simple greedy algorithm for recognizing partial 3-trees that computes such an elimination order in $O(n \log(n))$ time. Their algorithm maintains three queues for vertices of degree 1, 2, and 3, where a degree 3 vertex x is placed in the third queue only if local connectivity tests show that elimination of x would not increase the treewidth of the graph. The algorithm then eliminates any vertex from the first nonempty queue, and updates queues as degrees change. They prove that this algorithm finds an elimination sequence if, and only if, the graph is a partial 3-tree. The connectivity tests take logarithmic time, so determining an optimal vertex ordering takes $O(n \log n)$ time. Moreover, if the graph has treewidth $w \leq 3$, then each eliminated vertex has degree, at most, w.

For an interaction graph in which the maximum number of states for a vertex is $s = \max_v |\mathcal{S}_v|$, if dynamic programming is given an elimination sequence in which vertex degrees are at most w, it finds the optimum score in $O(ns^{w+1})$ time. This algorithm shows that optimizing hydrogen placement falls into the class of *NP*-hard problems whose instances admit polynomial-time solutions when their underlying graphs have constant treewidth [Bodlaender 1988; Arnborg and Proskurowski 1989; Arnborg et al. 1991]. For our special case of partial 3-trees, we can, therefore, observe:

THEOREM 4.2. *Dynamic programming can compute the optimum score of an n-vertex interaction graph with treewidth $w \leq 3$ in $O(ns^{w+1} + n \log(n))$ time, where $s = \max_v |\mathcal{S}_v|$ is the maximum number of states for any vertex.*

In practice, this algorithm can be extended to reduce graphs with treewidths greater than three to a manageable size; simply place the degree-3 vertices rejected by the connectivity tests into a fourth queue, and eliminate vertices from this queue if the other queues have been exhausted. When all queues are empty, then resort to brute force enumeration of the states for vertices that remain in the graph. As a heuristic, sort the vertices in the fourth queue by decreasing the number of states so that dynamic programming eliminates vertices with the most states first, with the aim of maximally reducing the size of the state space that will remain when brute-force enumeration begins.

Since the original publication of this conference paper, we have shown that another problem from structural biology, the side chain-placement problem, can also be captured by interaction graphs with small treewidth and optimized by dynamic programming [Leaver-Fay et al. 2005]. Cai et al. [Song et al. 2005] have shown how dynamic programming on interaction graphs can be used to solve protein threading problems.

5. IMPLEMENTATION

We have implemented both the decomposition of the dot-based scoring function into a hypergraph and the dynamic programming algorithm on a hypergraph in C++. We use one set of classes to represent a hypergraph for scoring and another set of classes to represent a hypergraph for performing dynamic programming. For each problem instance, we first compute the scores, in one phase, and then optimize the scores in a second phase.

We separate scoring and optimization into two phases so that we can score the hypergraph once and save the scores for multiple optimization runs; dynamic programming must be run several times on similar problems. The reason for these several rounds of dynamic programming is that REDUCE reports not only the optimal network configuration, but also for each flippable group, i, it reports the optimal network configuration with i fixed in its sub-optimal flip state. The difference between the scores of the optimal network configuration and the sub-optimal network configuration for i is the score for flipping i. REDUCE does not deem it worth flipping a group if the flipped state is only marginally better than the original—it instead defers to the decision of the crystallographer.

The original brute force force algorithm found these additional sub-optimal network states during its exhaustive search. Dynamic programming, however, does not exhaust the search space and can produce only a single optimal network configuration. To compute the score for flipping i, we perform an additional round of dynamic programming for i. These additional rounds are quite fast as each additional round relies on scores already computed for the original problem; no additional scoring is needed. That is, we score the network once and store those scores in a hypergraph. We solve the original optimization problem once to find the optimal network configuration. Then for each flippable group, we solve a smaller optimization problem to find the sub-optimal network configurations. For each of these optimizations, the sub-optimal vertex has half as many states as in the original optimization. ASN and GLN each have a single state when restricted to their sub-optimal flip state; HIS has three states (protonation at ND1, NE2, or both) when restricted to its sub-optimal flip state.

Our dynamic programming hypergraph is designed to fully optimize partial 3-trees. We, therefore, represent hyperedges of degree at most four. That is, the implementation decomposes the scoring function into one-, two-, three, and four-body interactions and then into a catch-all set of higher-body interactions. For interactions that depend on four or fewer vertices, the implementation scores all involved dots in the initial scoring phase and store those scores. If the implementation encounters dots that participate in a five-body interaction, then it "punts." It will not score those dots as part of the scoring hypergraph, but rather, wait to score them until during the brute-force enumeration

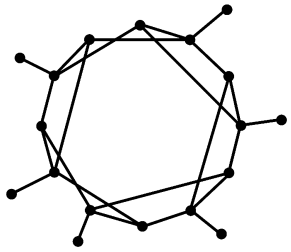


Fig. 4. 1OTF's large interaction graph.

phase that follows dynamic programming. The implementation induces a five-clique on vertices involved in the five-way interaction so that dynamic programming will not remove any of these five vertices before brute-force enumeration begins.

In addition to replacing the brute-force enumeration with dynamic programming, we have also converted all of the data structures in REDUCE to STL. Better memory management alone resulted in a sixfold performance gain. However, the orders-of-magnitude performance gain that we have observed comes from the combination of the scheme to decompose the scoring function and dynamic programming. It is always gratifying when an algorithm means that problems that were previously infeasible are now readily solved.

Version 3.02 of REDUCE includes our dot score decomposition algorithm and dynamic programming, and is now available from `http://kinemage.biochem.duke.edu/software/`.

6. EXPERIMENTS

We validated our implementation on a set of 2002 PDB files, containing 34,248 subproblems with nontrivial graphs. There were 86 subproblems for which the state spaces had over 100,000 states. The previous version of REDUCE (with STL) took 6 h 47 min to complete optimization of the remaining subproblems in this test set, avoiding optimization of these 86 large subproblems. Our dynamic programming / brute-force hybrid was able to optimize all subproblems, including the omitted 86, in 56 min. All runtime experiments were performed on a 2.5 GHz Mac G5 with 4 GB of RAM.

In our test set, we encountered one protein, 1OTF, with a problem instance whose graph had a treewidth greater than three. This problem instance was also the one with the largest state space size, 4.8×10^{12}. We have illustrated the problem instance's interaction graph in Figure 4. This problem did not contain any three-way mover overlap. Our strategy of eliminating as many degree-3 vertices as possible resulted in an irreducible core of six vertices with a state space size of 64. Brute-force optimization of this irreducible core proceeded rapidly.

Of the 34,247 other graphs in this test set, two were of treewidth-three, and the rest were of treewidth-one or-two. There was no mover overlap of higher

Table I. Large Interaction Graphs

PDB	|V|	|State Space|	TW	$\|H_3\|$	Effort	Time (sec)
2NLR	9	1.016×10^6	2	1	390	0.06
1HTP	7	1.143×10^6	2	0	675	0.09
1JDB	9	1.369×10^6	2	0	252	0.10
1A9X	9	1.493×10^6	2	1	252	0.12
1XVA	7	1.693×10^6	2	1	2246	0.09
1HVB	9	2.246×10^6	1	0	350	0.08
1EK6	8	2.281×10^6	2	2	1078	0.09
3PTE	9	2.546×10^6	1	0	370	0.08
1A8R	8	2.765×10^6	2	1	1844	0.13
1E4M	11	8.709×10^6	2	2	794	0.12
1H72	9	1.008×10^7	2	1	918	0.06
1JIL	12	1.739×10^7	1	0	242	0.04
1GM7	11	2.013×10^8	2	0	508	0.09
1A8R	11	2.395×10^8	2	2	1337	0.12
1PNK	11	3.137×10^8	2	0	650	0.10
1A8R	11	3.835×10^8	2	2	1198	0.12
1A8R	11	5.552×10^8	2	2	2262	0.47
1A8R	11	8.983×10^8	2	1	2290	0.17
1A8R	11	1.344×10^9	2	2	2191	0.19
1A8R	11	1.670×10^9	2	2	2758	0.19
1A8R	11	1.858×10^9	2	2	2216	0.18
1A8R	11	2.312×10^9	2	2	1963	0.15
1A8R	11	2.961×10^9	2	2	2704	0.48
1A8R	11	3.064×10^9	2	2	1974	0.16
1QRR	11	2.008×10^{10}	2	6	7317	0.98
1OTF	18	4.812×10^{12}	4	0	786	0.08

Interaction graphs from the 2002 PDB test set with more than one million network states. The column $|V|$ reports the number of vertices in each interaction graph. There were few cases of degree 3 overlap that required representation with hyperedges of degree 3 ($|H_3|$). Only one graph had a treewidth (TW) greater than 3. After dynamic programming eliminated all but 6 of the vertices from this graph, enumeration of their states completed the optimization. The *Effort*, reported in the second to last column, represents the number of state combinations examined during dynamic programming (and, for 1OTF, in the round of brute-force enumeration that followed). Profiling reveals that 0.1% of the running time is spent in dynamic programming; the running time is dominated by the precomputation of dot scores. GTP Cyclohydrase I (1A8R) contained several large connected components, and so it appears in several places in this table.

order than three. There was no four-way overlap, so whereas the degree-4 hyperedges proved unnecessary, our strategy of decomposing the scoring function into interactions of order-4 or less was sufficient to handle all observed input cases. We present a short list of the largest problems we encountered in this test set in Table I. The majority of these problems were treewidth-2, and could be solved without resorting to brute-force enumeration on a reduced problem. Our dynamic programming/brute-force hybrid is able to solve each subproblem in less than a second. The speedup that our improved algorithm represents can be measured by comparing for these large problems their state space sizes with the effort (in the second-to rightmost column of Table I) that dynamic programming required to optimize them. In the case of 1OTF, that speedup is almost ten orders of magnitude.

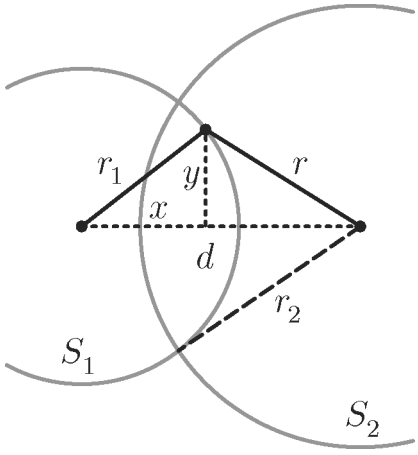


Fig. A1. Notation for two atom spheres.

APPENDIX

CONTACT DOTS FOR SCORING ATOMIC INTERACTION

Contact dots are motivated in Word et al. [1999a] as a discrete approximation to a continuous scoring function on the overlap volume, but they actually approach a function on surface area in an overlap region as the density increases. In the rest of this section, we derive both of these continuous functions and compare them to the discrete approximation. This has no effect on how we can speedup the computation in REDUCE, but this analysis may be important for extending contact dots to a more continuous scoring, based on Voronoi diagrams. It also shows the amount of approximation error the biochemists are willing to tolerate for this problem.

Assume that we are given the van der Waals spheres, S_1 and S_2, for a pair of nonbonded atoms. Since hydrogen-bond scores and overlap scores can be computed by the same function with a different multiplier, we look only at the overlap score for now.

Let us consider the score of a dot at (x, y) on a sphere S_1 centered at the origin of radius r_1 if that dot lies inside a sphere S_2 centered at $(d, 0)$ of radius r_2. By rotation and translation, we can bring any pair of atoms to such a configuration, as illustrated in Figure A1.

Word et al. [1999a] state that, "Hydrogen bonds and other overlaps are quantified by the volume of the overlap. Those volumes are easily measured by summing the spike length (l_{sp}) at each dot..." In fact, this is different from the volume. As the sampling density goes to infinity, we can express this as an integral of spike length over the surface of the overlap region.

If we denote the distance from (x, y) to $(d, 0)$ by r, then $x^2 + y^2 = r_1^2$ and $(d - x)^2 + y^2 = r^2$, so $x = (d^2 + r_1^2 - r^2)/(2d)$. Spike length for a dot inside sphere S_2 is one-half the distance to the boundary, which, in our notation, is $l_{sp} = (r_2 - r)/2$.

From geometry, the portion of sphere S_1 in a halfspace $X > x$ is a spherical cap whose surface area can be expressed as a function of r:

$$A(r) = 2\pi r_1(r_1 - x) = 2\pi r_1\left(r_1 - \frac{d^2 + r_1^2 - r^2}{2d}\right) = \frac{2\pi}{d} r_1(r^2 - (d - r_1)^2)$$

The derivative $A'(r) = 2\pi r_1 r/d$. Thus, the overlap calculation is the integral

$$\int_{d-r_1}^{r_2} ((r_2 - r)/2)A'(r)dr = \frac{\pi r_1}{d}\int_{d-r_1}^{r_2} r(r_2 - r)dr$$

$$= \frac{\pi r_1}{6d}\left(2(d - r_1)^3 - 3r_2(d - r_1)^2 + r_2^3\right)$$

The volume enclosed by the plane $X = x$ and the cap of sphere S_1 can be expressed as a function, $V_{r_1}(x) = (\pi/3)(x^3 - 3r_1^2 x + 2r_1^3)$. The plane $X = (d^2 + r_1^2 - r_2^2)/(2d)$ contains the intersection $S_1 \cap S_2$, and partitions the overlap volume into regions bounded by two spherical caps. Thus, the total overlap volume is

$$\begin{aligned} V &= V_{r_1}\left(\frac{d^2 + r_1^2 - r_2^2}{2d}\right) + V_{r_2}\left(\frac{d^2 + r_2^2 - r_1^2}{2d}\right) \\ &= \frac{\pi}{12d}\left(d^4 - 6d^2\left(r_1^2 + r_2^2\right) + 8d\left(r_1^3 + r_2^3\right) - 3\left(r_1^2 - r_2^2\right)^2\right) \end{aligned}$$

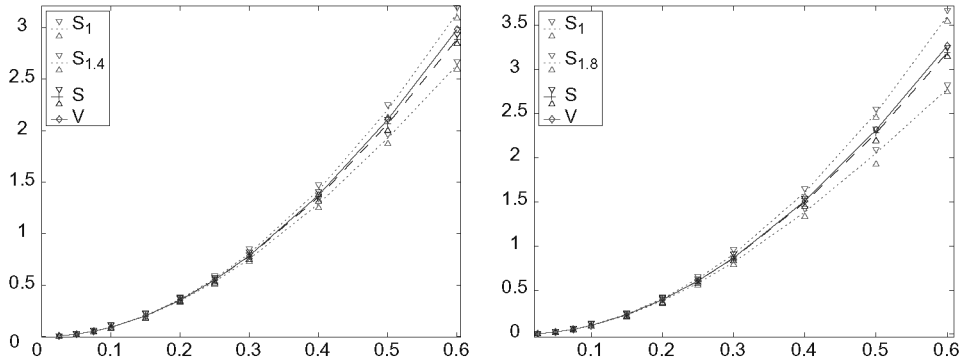
We could choose to assess sphere S_1 either one-half the total volume, $V/2$, the volume of its cap, $V_{r_1}((d^2 + r_1^2 - r_2^2)/(2d))$, or the portion of volume on the side of the bisector between spheres S_1 and S_2 that is closest to S_2, which is a more complicated formula, but is closest to the aim of the dot approximation. These three quantities are identical when the van der Waals radii are the same for the two atoms.

The surface integral and dot approximation are reasonably close to the volume score when a contact is measured from both sides, but can deviate when the contact is scored on only one side for atoms whose radii differ.

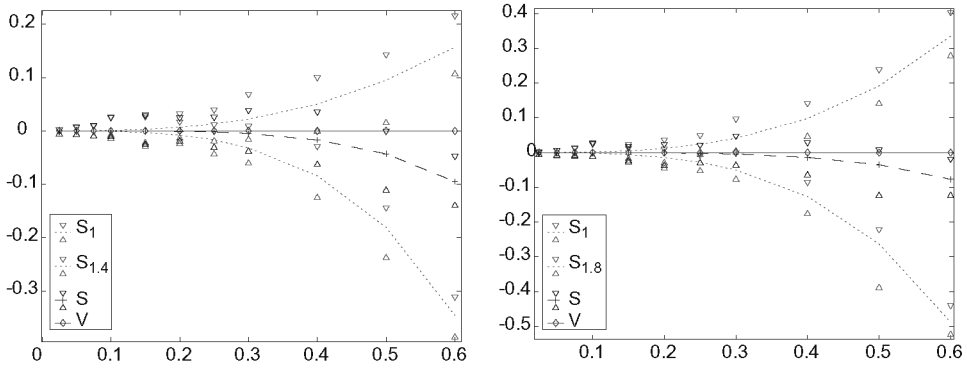
The dot scores depend on precisely how the sample dots lie relative to neighboring spheres. Even while maintaining distance d and penetration distance $d - r_1 - r_2$, rotating the spheres can change the scores as different number of dots enter overlap configurations. We used MATLAB's `fminsearch` to determined the minimum and maximum scores from about 30 initial starting positions distributed on the sphere of directions.

Figure A2 displays graph of the overlap scores for sphere of radius 1 overlapping a sphere of radius 1.4 (left) and 1.8 (right). It also displays absolute and relative error of dot and surface scores to the volume score. The x axis in these plots is the depth of penetration $d - r_1 - r_2$, which ranges from 0 to 0.6. Thus, $2.4 \geq d \geq 1.8$ on the left and $2.8 \geq d \geq 2.2$ on the right.

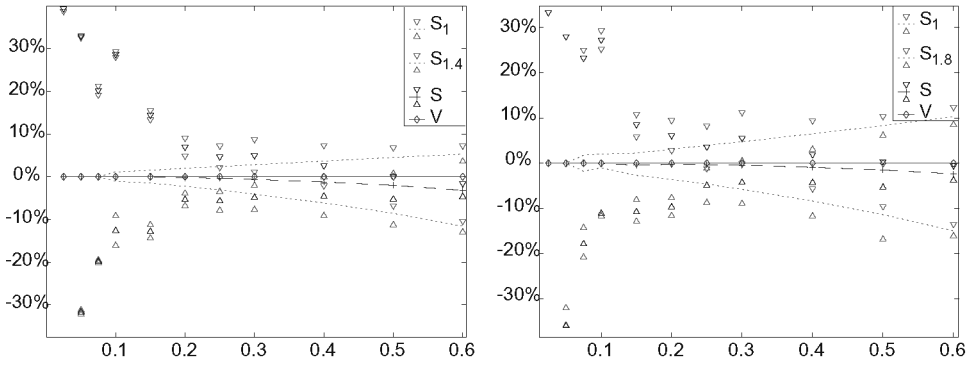
In each graph, the volume score (V) is a solid line marked with diamonds. Dashed and dotted lines are the surface score, which is the limit of the dot scores as the number of dots increases without bound. Each line has pairs of upper



Overlap score vs. penetration depth d - r_1 - r_2 for r_1 =1 and r_2 =1.4 or 1.8



Absolute error of surface score to volume vs. penetration depth d - r_1 - r_2 for r_1 =1 and r_2 =1.4 or 1.8



Relative error of surface score to volume vs. penetration depth d - r_1 - r_2 for r_1 =1 and r_2 =1.4 or 1.8

Fig. A2. Overlap scores and errors as functions of the penetration distance $d - r_1 - r_2$ for a sphere of radius 1 overlapping a sphere of radius 1.4 (*left*) and 1.8 (*right*). The volume score (V) is a solid line marked with diamonds. Dashed and dotted lines are the limit of the dot scores with infinite dot density; upper and lower triangles mark the highest and lowest dot scores returned by REDUCE for these radii and penetration depths. There are three sets of dotted lines and triangles, since the score can be computed from the larger atom (e.g., $S_{1.8}$), the smaller atom (S_1), or the average of these two (S).

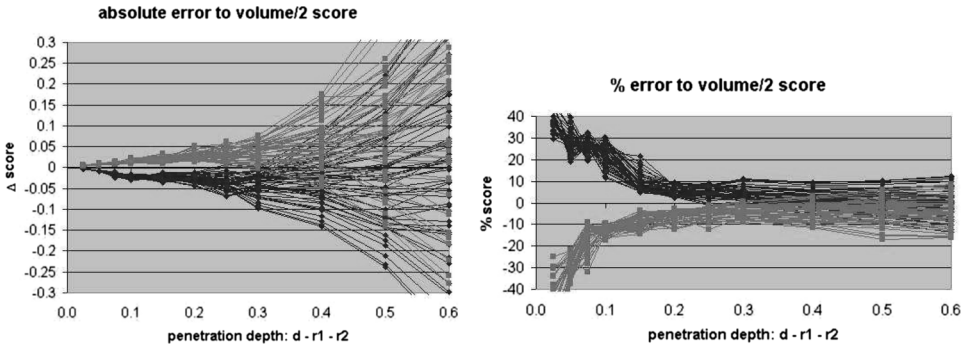


Fig. A3. Absolute and relative volumes for all pairs of radii.

and lower triangles, which mark the highest and lowest dot scores returned by REDUCE for these radii. For each pair of different radii, the score can be computed from the larger atom (e.g. $S_{1.8}$), the smaller atom (S_1), or the average of these two (S).

The lowest graphs show that discretization gives high relative error on the small overlaps, but since the score is small (top) and differences are small (middle) we can guess that this is probably not significant for evaluation of dot scores. Averaged surface score (S) tracks the volume score (V) well. The larger or smaller atom scores, which are relevant only for atoms that are scored from one side, show greater deviation from the volume score. In fact, these consistently over-or underestimate the volume, depending on whether we compute on the side of the smaller or larger atom.

If we had added the other options for volume scores (cap volume, or bisected volume) they would be outside of the dotted lines for surface scores. For example, when the radii are 1 and 1.4, then the cap volumes are within $\pm$16–20% of $V/2$, and the bisected volume are within $\pm$8–14% than $V/2$. The exact surface scores, S_1 and $S_{1.4}$ are within $\pm$1–8%. Thus, we did not include comparisons with these other volume scores, although they are interesting options for continuous, Voronoi-based scoring.

Figure A3 shows that the dot scores for other radii have similar behavior. It plots absolute and relative errors for approximating volume score by the dot scores from atoms whose radii take on all pairs of values from [1, 1.17, 1.4, 1.55, 1.65, 1.75, 1.8]. Min and max error are plotted for each ordered pair of radii.

ACKNOWLEDGMENTS

We would like to thank Professors Dave and Jane Richardson, Michael Word, Ian Davis, and Bryan Arendall for their tremendous help and enthusiasm. We would also like to thank the reviewers for their keen insights and thoughtful comments, and to thank the editors for organizing this special issue. This research has been partially supported by NSF grant 0076984 and NIH grant GM–074127.

REFERENCES

ARNBORG, S., LAGERGREN, J., AND SEESE, D. 1991. Easy problems for tree-decomposable graphs. *J. Algorithms 12*, 2, 308–340.

ARNBORG, S. AND PROSKUROWSKI, A. 1986. Characterization and recognition of partial 3-trees. *SIAM Journal of Algorithms and Discrete Methods 7*, 305–314.

ARNBORG, S. AND PROSKUROWSKI, A. 1989. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics 23*, 11–24.

BODLAENDER, H. L. 1988. Dynamic programming on graphs with bounded treewidth. In *Proc. 15th Int. Colloq. Automata, Languages and Programming*. Springer Verlag, Lecture Notes in Computer Science 317, Tampere, Finland, 105–118.

BRANDEN, C. AND TOOZE, J. 1999. *Introduction to protein structure*, Second ed. Garland Publishing.

DESMET, J., MAEYER, M. D., HAZES, B., AND LASTERS, I. 1992. The dead-end elimination theorem and its use in protein side-chain positioning. *Nature 356*, 539–541.

LEAVER-FAY, A., KUHLMAN, B., AND SNOEYINK, J. 2005. An adaptive dynamic programming algorithm for the side chain placement problem. In *Pacific Symposium on Biocomputing, 2005*. World Scientific, The Big Island, HI, 17–28.

LOVELL, S. C., DAVIS, I. W., III, W. B. A., DE BAKKER, P. I. W., WORD, J. M., PRISANT, M. G., RICHARDSON, J. S., AND RICHARDSON., D. C. 2003. Structure validation by c-alpha geometry: phi, psi, and cβ deviation. *Proteins: Structure Function and Genetics 50*, 437–450.

RICHARDSON, D. C. AND RICHARDSON, J. S. 2001. *Mage, probe, and kinemages.* Vol. F. Kluwer Publishers, Dordrecht.

RICHARDSON, J. S. AND RICHARDSON, D. 1987. Some design principles: betabellin. In *Protein Engineering*, D. Oxender and C. Fox, Eds. Alan R. Liss, Inc., New York, 149–163, 340–341.

SONG, Y., LIU, C., HUANG, X., MALMBERG, R. L., XU, Y., AND CAI, L. 2005. Efficient parameterized algorithm for biopolymer structure-sequence alignment. In *Workshop on Algorithms in Bioinformatics, 2005*. Springer-Verlag, Mallorca, Spain, 376–388.

WORD, J. M., JR., R. C. B., PRESLEY, B. K., LOVELL, S. C., AND RICHARDSON, D. C. 2000. Exploring steric constraints on protein mutations using mage/probe. *Protein Sci 9*, 2251–2259.

WORD, J. M., LOVELL, S. C., LABEAN, T. H., TAYLOR, H. C., ZALIS, M. E., PRESLEY, B. K., RICHARDSON, J. S., AND RICHARDSON, D. C. 1999a. Visualizing and quantifying molecular goodness-of-fit: Small-probe contact dots with explicit hydrogen atoms. *Journal of Molecular Biology 285*, 1711–1733.

WORD, J. M., LOVELL, S. C., RICHARDSON, J. S., AND RICHARDSON, D. C. 1999b. Asparagine and glutamine: Using hydrogen atom contacts in the choice of side-chain amide orientation. *Journal of Molecular Biology 285*, 4, 1735–1747.

Received May 2004; revised June 2006; accepted December 2006