Faster placement of hydrogens in protein structures by dynamic programming

Andrew Leaver-Fay Yuanxin Liu Jack Snoeyink

Abstract

Word and coauthors from the Richardsons' 3D Protein Structure laboratory at Duke University propose dot scores to measure inter-atomic interactions in molecular structures. They use these scores in their program REDUCE, which searches for the optimal placement of hydrogen atoms in molecular structures from the Protein DataBank (PDB). We investigate the accuracy and computation of these scores. By replacing part of their search by a dynamic programming algorithm based on partitioning the interaction graph between amino acid residues, we observe a speed-up of up to seven orders of magnitude.

1 Introduction

In molecular biology, it is frequently important to analyze atom contacts within a protein molecule to accurately determine and evaluate its structure. As tools for atom contact analysis, Word and others from David and Jane Richardsons' 3D Protein Structure laboratory [2, 3, 4, 5, 6] have developed a set of programs, all of which are open-source and can be directly used over the internet at `http://kinemage.biochem.duke.edu`. One of these programs, named REDUCE, is used to add hydrogens to a Protein DataBank (PDB) molecular structure based on atom contacts [6]. It uses a brute-force search for the hydrogens placement that minimizes an energy score function.

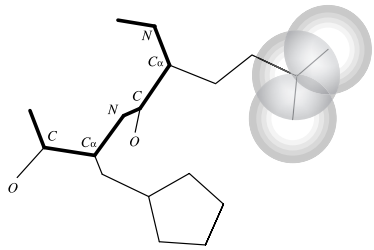


Figure 1: Protein backbone & side chain bonds with three van der Waals spheres

REDUCE's brute-force search is, in fact, reasonably fast for the majority of the input cases encountered. Unfortunately, a number of "bad" input cases noted by the computational biologists cannot be computed within acceptable time. We found a dynamic programming formulation of the optimization problem which succeeds for low treewidth graphs, implemented our algorithm, and replaced their search subroutine with ours. We compared the modified version of REDUCE with the original, using the "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. This "sequence" is the sequence of amino acids, which can be identified by their names or by their three letter abbreviations. There are 20 different amino acids which nature uses to build its proteins. Figure **??** shows two amino acids in a portion of a protein. The backbone of the protein is the composed of a repeating sequence of $-N-C_\alpha-C-$ atoms. Each amino acid contributes one unit of this repeat. The 20 amino acids differ in the chemical groups bound to the C_α atom. A protein's sequence refers to the side chain composition but it is the combination of the side chain and backbone chemical structure that determine it's physical structure, its "fold."

A popular model for proteins is the hard-sphere molecular model: an atom is modelled as a sphere in 3D centered at the atom's coordinate and with radius equal to the atom's van der Waals radius. Spheres may overlap only for atoms that share a chemical or hydrogen bond. Biologists like this simple geometric model of proteins not only because it makes visualization easy but also because the interaction between protein atoms can be modelled reasonably well as collisions between spheres.

As mentioned above, the Richardsons' 3D Protein Structure laboratory has developed a set of software tools that all take a protein model as input and analyze the protein's atom interactions based on their model as spheres. On the "front end", a program named KINEMAGE visualizes a protein model and shows colored dots and spikes for atom contacts between non-

bonded atoms. The dots and spikes are produced by a program named PROBE. Finally, there is 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 coordinates of the atoms specified in angstroms and the bonds between the atoms. The most popular method for structure determination, X-ray crystallography, is based on electron density. Unfortunately, due to technology limitations, electron density is sufficiently resolved for only the heavy atoms such as C, N and O. Hydrogen atoms, though roughly half of the atoms in any given protein, are not visible in an X-ray diffraction experiment.

REDUCE takes a protein model from a PDB file and finds the best placement of hydrogens onto the model. The output of REDUCE can then be used by PROBE and KINEMAGE to analyze the interaction of atoms participated by hydrogen atoms—which the biologists have considered to be important [3, 4]. The broad goal of REDUCE, when searching for the best hydrogen placement, is to minimize "clashes" between atoms—that is, when two atoms van der Waals spheres overlap in space—and to maximize the amount of hydrogen bonding. A hydrogen bond is formed as an overlap between a hydrogen atom and a polar heavy atom to which the hydrogen is not chemically bound. Not all hydrogen atoms can participate in hydrogen bonds - only those chemically bound to polar heavy atoms. The heavy atom not chemically bound to the hydrogen is called the *acceptor*. The hydrogen bond *donor* refers sometimes to the other, chemically bound, heavy atom, and sometimes to the hydrogen. In Appendix A, we refer to the hydrogen when we use the term hydrogen bond donor.

What choices does REDUCE have to add hydrogens? The positions of hydrogen atoms bound to most heavy atoms are fixed. REDUCE computes these positions with simple vector geometry. Other hydrogens belong to OH, SH or NH_3^+ groups that have rotational freedom. The rotation is commonly sampled at fixed intervals to make the choices discrete. Finally, there are certain side chains —ASN, GLN, and HIS, in particular— that appear symmetric in two different "flip" positions to the structure determination technology unless hydrogens are placed on them. This is illustrated in Figure 2 and Figure 3. Unless the structural biologist depositing the protein to the PDB has paid careful attention to where the hydrogens must lie, the structure is likely not correct. By explicitly modelling the hydrogen atoms for these groups, REDUCE is able to resolve structural ambiguity for heavy atoms as well.

REDUCE formalizes the search for possible hydrogen placement choices as an assignment of states to a collection of movers. A *mover* is a group of atoms that can be rotated or "flipped" together to change the coordinates of the hydrogens in the group. Each choice of rotation or flip defines a state for a mover.

The "best" hydrogen placement is defined to be the configuration of movers that minimizes an energy function. The energy function inspects all the nonbonded atom contacts—an atom contact is made whenever two atoms' van der Waals spheres overlap in space. When a contact is made by two atoms that form a donor/acceptor pair for a hydrogen bond, the overlap volume is rewarded by a negative energy score, otherwise, it is penalized by a positive energy score. Specifically,

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

In theory, the constants are chosen to give "an overall scoring profile similar in shape to the van der Waals function for an isolated pairwise interaction" [5]. The overlap volume is approximated with dots sampled on the surface of the overlapping atoms weighted by their penetration depth. Interestingly, this approximation does not converge to the overlap volume as the sampling density increases, even though the biologists are satisfied with its accuracy. We present a detailed analysis of this scoring function in Appendix A.

To minimize the energy score, the movers are first grouped into sets, called "cliques" that are isolated in their atom contact interactions. To be more formal, we define an interaction graph with a vertex for each mover in the clique and an edge between two mover vertices if there exist some states of the movers that induce contacts between their atoms. A clique is a set of movers corresponding to the vertices of a connected component of the interaction graph. Clearly, minimizing the energy score for each clique also minimizes the total energy score. The minimization for each clique is done by brute-force enumeration of all possible combination of the movers' states.

3 Dynamic Programming

Section 2 mentioned the interaction graph that REDUCE uses to find atom groups, called cliques, so that all contact interactions occur within cliques. REDUCE discards this interaction graph when searching for a minimum score for a clique. Instead, we partition this

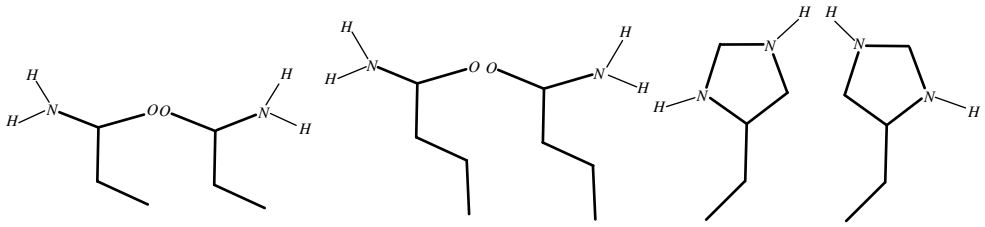


Figure 2: From left to right: two flip states for each of three amino acids: ASN, GLN and HIS

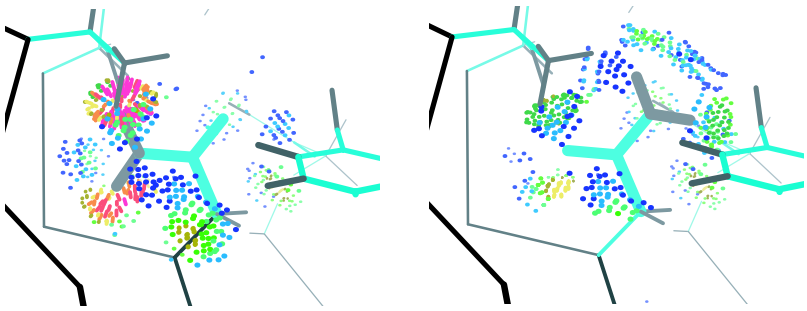


Figure 3: ASN induces different atom contacts in two "flip" positions. Spikes (red) indicate sphere overlaps, while dots pillows (green) indicate hydrogen bonds. (Images produced by KINEMAGE program)

interaction graph and apply dynamic programming to compute the score.

First, however, we need to extend the interaction graph so that it captures not only pairwise atom contact interactions but also interactions among more than two atoms. Specifically, define a hypergraph $G = (V, E)$. Each vertex in V represents a mover. Each hyperedge in E is a subset of V; exactly which subsets are hyperedges is defined in the rest of this paragraph. We say there is a possible *complete interaction* between a set of movers if there exist some states for the movers such that we can choose an atom from each mover, intersect their van der Waals spheres, and discover that the intersection is non-empty. A set of vertices $V' \subseteq V$ is a hyperedge if and only if there is a possible complete interaction among the movers represented by V'. We should note that the collection of hyperedges, E, in fact is an abstract simplicial complex. That is, if e is a hyperedge, than any set of vertices $e' \subseteq e$ is also a hyperedge. We should also note that a set containing a single vertex is a hyperedge by definition.

With this interaction graph, we can decompose the scoring function. Given a set of state assignments to the movers, the score is the sum of the score for each dot. The score of a dot can either be determined by its interaction with another mover or with background stationary atoms. We can re-distribute the summation of dot scores to the hyperedges as the following: for each dot score, take the maximal set of movers whose van der Waals spheres contain the dot. These movers identify a hyperedge in the interaction graph which we distribute the dot score to. In the case when a dot from a mover represented by vertex v is not contained in any other mover's spheres but is contained in some sphere from a stationary atom—and therefore still contributes a score, we distribute the dot's score to the hyperedge $\{v\}$. The score can now be described as the sum of the scores for the hyperedges. We can summarize this a little more formally as the following. Let d denote a dot, $\{d\}$ the set of all dots, $score(d)$ the score for a dot, and $d \rightarrow e$ a shorthand for the statement that the dot is distributed to the hyperedge.

$$\text{total score} = \sum_{d \in \{d\}} score(d) = \sum_{e \in E} (\sum_{d \rightarrow e} score(d))$$

Given a hyperedge e and an assignment of states to its vertices, we now have a definition of its score as $\sum_{d \rightarrow e} score(d)$.

With the decomposition of the total score of a clique into the sum of its hyperedge scores, we are ready to decompose our optimization problem into subproblems as well. Let us first formally define the problem.

Problem: Given

- a hypergraph $G = (V, E)$,
- an associated set of states $\mathcal{S}(v)$ for each vertex $v \in V$, and
- a scoring function f_e for each hyperedge $e \in E$, defined as a mapping $f_e : \prod_{v \in e} \mathcal{S}(v) \rightarrow \Re$ that

returns score for an assignment of states to the vertices in e,

find the assignment of states to all vertices such that the sum of the hyperedge scores is minimized.

We also introduce the following notation for convenience.

- For a set of vertices V', let $\mathcal{S}(V') = \prod_{v \in V'} \mathcal{S}(v)$ denote the state space of the vertices 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 vertices V', their state vector $S_{V'}$, we let $S_{V'' \subseteq V'}$ denote the partial state vector just for a subset of V', V''.

The way this problem can be decomposed into subproblems depends on the connectivity of the graph. Suppose we can find a *vertex cut*, A, of the graph. That is, the rest of the vertices, $V - A$, can be partitioned into two sets, L and R, such that no hyperedges exist between them(See Figure 4). Equivalently, the hyperedges can be partitioned into three sets:

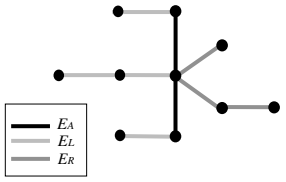


Figure 4: Partition the hyperedges into three sets.

- E_A, the hyperedges that are subsets of A.
- E_L, the hyperedges that are subsets of $A \cup L$ but not in E_A.
- E_R, the hyperedges that are subsets of $A \cup R$ but not in E_A.

Let $T(E, S_V)$ denote the minimum score for a collection of hyperedges E with some of its vertices V fixed in states S_V. Then, the minimum score for the graph is

$$\min_{S_A \in \mathcal{S}(A)} T(E_L, S_A) + T(E_R, S_A) + \sum_{e \in E_A} f_e(S_{e \subseteq A})$$

This formula illustrates the essential idea of how the interaction graph can help us decompose and combine subproblems. We now show a recursive formula which can be implemented as a bottom-up dynamic program algorithm.

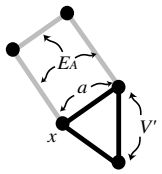


Figure 5: Identify an edge set by its vertex cut a.

In order to make the recursive formula concise, let us first name some objects in the graph.

- V_f, a set of vertices that we want to fix in states S_{V_f}
- a vertex x such that $A = V_f \cup \{x\}$ cuts the graph into multiple subgraphs. That is, the rest of the vertices, $V - A$, can be partitioned into multiple sets of vertices such that no hyperedges exist between them. Equivalently, the hyperedges can be partitioned into sets of two kinds.
  - E_A, the hyperedges that are subsets of A.
  - for each vertex partition V', we have a set of hyperedges that are subsets of $A \cup V'$ but not in E_A. This hyperedge set can actually be identified by a set of vertices $a \subseteq A$ which cut V' from the rest of the vertices. We therefore denote the hyperedge set identified by its vertex cut a as $E_{\times a}$ (See Figure 5), and we denote the collection of vertex cuts by $\{\times\}$.

Then, the minimum score for a set of hyperedges E with fixed vertex states S_{V_f} can be recursively defined as the following.

$$T(E, S_{V_f}) = \min_{S_A \in \mathcal{S}(x) \times S_{V_f}} \sum_{a \in \{\times\}} T(E_{\times a}, S_{a \subseteq A}) + \sum_{e \in E_A} f_e(S_{e \subseteq E_A})$$

The recursion terminates when we have a single hyperedge. That is,

$$T(e, S_e) = f_e(S_e)$$

The recursion can be implemented as a bottom-up dynamic programming algorithm as the following. For data structures, we keep a hypergraph G' to store solution scores to the subproblems. G' is initialized to be the interaction graph G but will have its vertices and hyperedges removed, as well as hyperedges added, by the process. We keep the following invariant.

- A hyperedge a in G' is either a hyperedge of G or a vertex cut that identifies a portion of the graph that was removed. Furthermore, let $E_{\times a}$ denote the hyperedges of the removed graph. We assert that a also stores the set of solution scores for the subproblems on $E_{\times a}$, $\{T(E_{\times a}, S_a), \forall S_a \in \mathcal{S}(a)\}$.

To initialize the process, set $G' = G$.

At each step, the following steps are performed.

1. Choose a vertex x and identify the set of vertices V' which x is connected to (by hyperedges).
2. Add a new hyperedge $e = V'$ if V' is not already a hyperedge. Then for each $S_e \in \mathcal{S}(e)$, compute $T(E_{\times e}, S_e)$ and store the result at e. Note that when we apply the recursive formula to compute $T(E_{\times e}, S_e)$, each recursive part of the right hand side must have already be stored on a hyperedge, according to the invariant. This step also maintains the invariant on the hyperedge e.

When we remove the last vertex x, we have a set of scores, each denoting the minimum of the total score when x is in some fixed state. Therefore, we report the minimum of these scores as the solution.

We have now completed describing a dynamic programming algorithm for computing the minimum score of a clique. Our implementation of the dynamic programming algorithm, however, is a more restricted version. We choose vertices for removal from G' which are connected to at most two other vertices in the graph. It can be easily shown that we can always succeed in finding such a vertex if and only if the graph has *treewidth* at most two. (Treewidth is a well-studied graph property. For a complete review of treewidth and algorithms on graphs of limited treewidth, please see [1]). If such a vertex cannot be found, however, our implementation applies brute force minimization on the rest of the interaction graph.

To give a few more details of our implementation, in order to store the subproblem scores on a hyperedge, we let each vertex—or a hyperedge containing one vertex—keep a score list indexed by the vertex's states. We let each edge keep a score table indexed by the state pairs of the edge's end-point vertices. We also observe that a hyperedge score is computed by summation of hundreds of dots—a considerable overhead if the computation is to be repeated. Therefore, we pre-compute these scores and store them either on the vertex's score list or on an edge's score table.

4 Experiments

We implemented the dynamic programming algorithm in C++ to integrate with the existing source for REDUCE. The program runs on an SGI platform as well as the Mac. Our goal was to make minimal modifications to the original source and we confined our substantial changes to within two subroutines: the one that performed the exhaustive search and the one that scored individual dots. We also made a small change to the implementation of one of their data structures to reduce, which alone resulted in a six-fold performance increase.

To test the performance gain, we executed our code for a small set of PDB files. For these files, the old version of REDUCE either took too long to optimize the cliques or had decided not to try. The PDB codes for these proteins were 1GCA, 1SVY, 4XIS, 1A7S 1SBP, and 1QRR, displayed in Figure 6. We present the running time of the entire program, running on a 300 `MHtz` SGI machine under similar processor loads, in Table 1. The network for clique 1QRR had over 1 billion possible states, and we accordingly present only a projected running time. It could be noted that while the time running the previous program was almost entirely spent optimizing these large cliques, our program spends most of its time elsewhere, and our performance gain for clique optimization alone is even better.

	old running time (t)	new (t')	$\log_{10}(\frac{t'}{t})$
1GCA	169.54 sec.	10.05 sec.	1.23
1SVY	142.67 sec.	4 sec.	1.55
4XIS	916.03 sec.	10.75 sec.	1.93
1A7S	1312.11 sec.	8.1 sec.	2.21
1SBP	4.5 hours	9.63 sec.	3.23
1QRR	6.5 years	22.26 sec.	6.95

Table 1: running time comparison

We also have tested our program against a larger set of 100 PDB files containing 677 cliques. Of these, there are 6 that show simultaneous overlap of three movers and which require a hyperedge of degree three to be accurately modelled. With minimal effort, we can handle these cases in conjunction with the current implementation of the dynamic programming algorithm by simply scoring their three-way interaction separately after having reduced the rest of the graph. None of the cliques showed any four-way overlap.

A final observation could be made about the treewidth of the interaction graphs we observed. With the exception of the few degree-3 hyperedge containing cliques mentioned above, all of the interaction graphs were of hyperedge-degree and treewidth of two or less. Thus the our implementation of the dynamic programming algorithm which handles exactly these interaction graphs is quite appropriate.

The main programming challenge has been in working with the dot- based scoring function. While this function is not in principle pairwise decomposable, we know we are able to get away with a degree 2 hypergraph due to the very low frequency of more complex networks. The existing code that surrounds the scoring function, however, does not lend itself to being picked

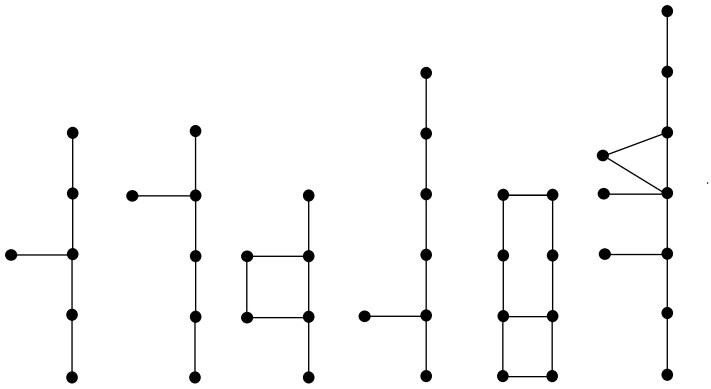


Figure 6: The interaction graph for six cliques.

apart, and forces complex logic in the innermost loop.

5 Future Work

There are several issues which remain at the conclusion of this paper. First, the optimization that is being computed by the original version and the current version we have implemented is not actually finding the optimal configuration for the entire molecule. When evaluating the interaction between two movers, there are two sets of dots that get scored: mover 1's dots inside mover 2 and mover 2's dots inside mover 1. When scoring a mover against the background, the mover's dots inside the background are counted, but the background's dots are not counted. This skews the optimization function to prefer good mover-mover interactions over mover-background ones in selecting the optimal state.

Second, we would like to increase the discretization level to make more fine-grained choices. After completing brute force optimization of each clique, the old version of reduce performs a "local optimization" step for the rotatable movers using 1 degree rotations near the "optimal" state. While this step generally reduces the energy score further, it is clearly missing the optimal solution were these fine grain discretizations available in the brute force optimization. With the performance gains we have achieved, it makes sense to include this local optimization in the global optimization step.

Finally, the scoring technique must be adjusted. As discussed in detail below, the volume approximation being performed in the previous and current version of REDUCE is not accurate. The actual dot density placed on the surface of the van der Waals spheres that is achieved is not what is sought, and thus the volume approximation is significantly off. This absolutely should be corrected for the next version of REDUCE. Also, the surface integral being estimated by the dots does not converge to the actual volume of overlap, though it is volume the biochemists sought to approximate. We would like to use this second source of error within the scoring scheme as a justification to change the way in which energy scores are evaluated, moving instead to an exact, Voronoi-like scoring scheme. Due to the high cost of the dot-based scoring function, we expect not only to see accuracy improvements, but also performance ones as well.

6 Acknowledgements

We would like to thank Professors Dave and Jane Richardson, Michael Word for their help. This research has been partially supported by NSF grant 0076984.

A Contact dots for scoring atomic interaction

Word et al. [5, 6] define contact dots and describe their use to visualize and score interactions between atoms in a protein model. 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 bonded to A are discarded. Each remaining dot p on A finds the closest van der Waals sphere B of an atom not bonded to A. This distance is considered negative if p is inside the van der Waals sphere B. Dot p is assigned favorable (i.e. negative) *contact energy* if it is outside of B, but within a small probe radius. Dot p is assigned favorable *hydrogen bond energy* if it is inside B (but not too deeply), and the atoms A and B are a hydrogen bond donor and acceptor. Finally, dot p is assigned an unfavorable (i.e. positive) overlap penalty if p lies inside B and either A,B is not a hydrogen bond donor/acceptor pair, or p is too deeply inside B. These contacts are easily visualized by drawing dots with favorable energies in cool blues and greens, and unfavorable overlaps with spikes of pink and red.

For applying dynamic programming in the next section, the key feature of contact dot scoring is its decomposability. Each dot can independently determine whether it is contained in a bonded atom, or what is the closest van der Waals sphere, and from that calculate its contribution to the total energy.

Contact dots are motivated in [5] 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 speed up 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 non-bonded atoms. Since hydrogen bond and overlap scores can be computed by the same function with a different multiplier, we look only at the overlap score for now.

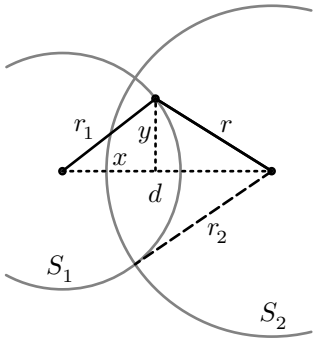


Figure 7: Notation for two atom spheres

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 7.

Word et al. [5] 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 $x2 + y2 = r_1 2$ and $(d - x)2 + y2 = r2$, so $x = (d2 + r_1 2 - r2)/(2d)$. Spike length for a dot inside sphere S_2 is 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:

$$A(r) = 2\pi r_1(r_1 - x) = 2\pi r_1(r_1 - \frac{d2 + r_1 2 - r2}{2d}) = \frac{2\pi}{d} r_1(r2 - (d - r_1)2).$$

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

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

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

The volume enclosed by the plane $X = x$ and the cap of sphere S_1 can be expressed as a function, $V_{r_1}(x) = (\pi/3)(x3 - 3r_1^2 x + 2r_1 3)$. The plane $X = (d2 + r_1 2 - r_2 2)/(2d)$ 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

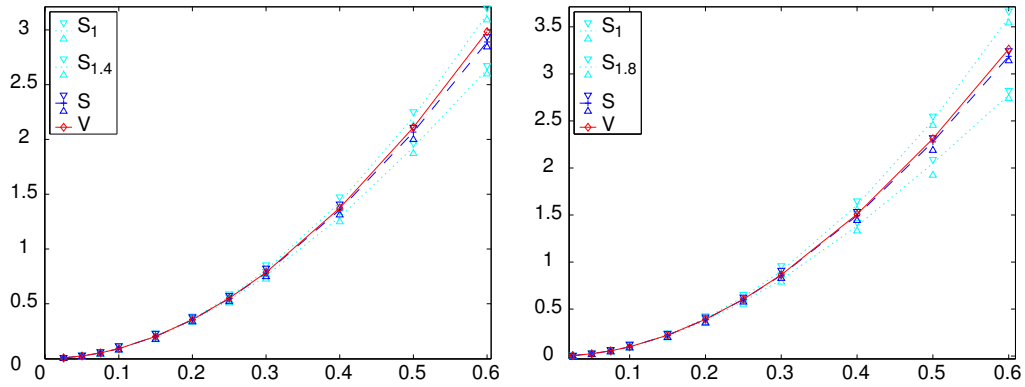
$$\begin{aligned} V &= V_{r_1}(\frac{d2 + r_1 2 - r_2 2}{2d}) + V_{r_2}(\frac{d2 + r_2 2 - r_1 2}{2d}) \\ &= \frac{\pi}{12d}\big(d4 - 6d2(r_1 2 + r_2 2) \\ &\quad +8d(r_1 3 + r_2 3) - 3(r_1 2 - r_2 2)2\big). \end{aligned}$$

We could choose to assess sphere S_1 either half the total volume, $V/2$, the volume of its cap, $V_{r_1}((d2 + r_1 2 - r_2 2)/(2d))$, 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.

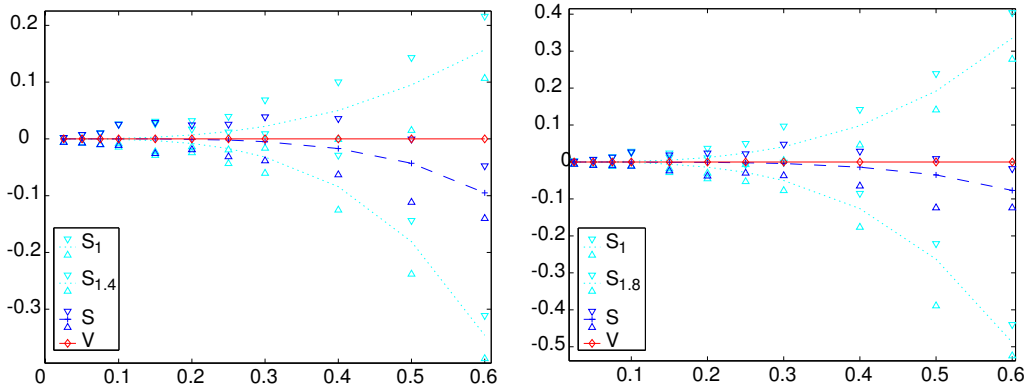
The surface integral and dot approximation are reasonably close to the volume score when a contact is measured from both sides, but can deviate when the contact is scored on only one side for atoms whose radii differ.

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.

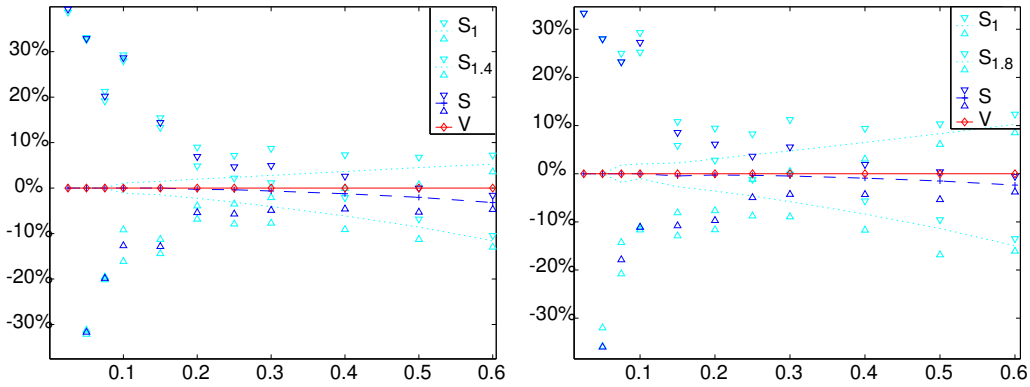
Figure 8 displays graph of the overlap scores for sphere of radius 1 overlapping a sphere of radius 1.4



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

Figure 8: 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).

(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 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 under estimate 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 ±16–20% of $V/2$, and the bisected volume are within ±8–14% than $V/2$. The exact surface scores, S_1 and $S_{1.4}$ are within ±1–8%. Thus, we did not include comparisons with these other volume scores, although they are interesting options for continuous, Voronoi-based scoring.

Figure 9 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.

References

[1] Hans L. Bodlaender. A tourist guide through treewidth. Technical Report 1992, Dept. Comput. Sci., Utrecht Univ.

[2] Simon C. Lovell, Ian W. Davis, W. Bryan Arendall III, Paul I. W. de Bakker, J. Michael Word, Michael G. Prisant, Jane S. Richardson, and David C. Richardson. Structure validation by c-alpha geometry: phi, psi, and c-beta deviation. *Proteins: Structure, Function, and Genetics*, 50:437–450, 2003.

[3] D. C. Richardson and J. S. Richardson. Mage, probe, and kinemages. In M.G. Rossmann and E. Arnold, editors, *International Tables for Crystallography*, volume F, pages 727–730. Kluwer Publishers, Dordrecht, 2001.

[4] J. M. Word, R. C. Bateman Jr., B. K. Presley, S. C. Lovell, and D. C. Richardson. Exploring steric constraints on protein mutations using mage/probe. *Protein Sci*, 9:2251–2259, 2000.

[5] J. Michael Word, Simon C. Lovell, Thomas H. LaBean, Hope C. Taylor, Michael E. Zalis, Brent K. Presley, Jane S. Richardson, and David C. Richardson. Visualizing and quantifying molecular goodness-of-fit: Small-probe contact dots with explicit hydrogen atoms. *Journal of Molecular Biology*, 285:1711–1733, 1999.

[6] J. Michael Word, Simon C. Lovell, Jane S. Richardson, and David C. Richardson. Asparagine and glutamine: Using hydrogen atom contacts in the choice of side-chain amide orientation. *Journal of Molecular Biology*, 285(4):1735–1747, 1999.

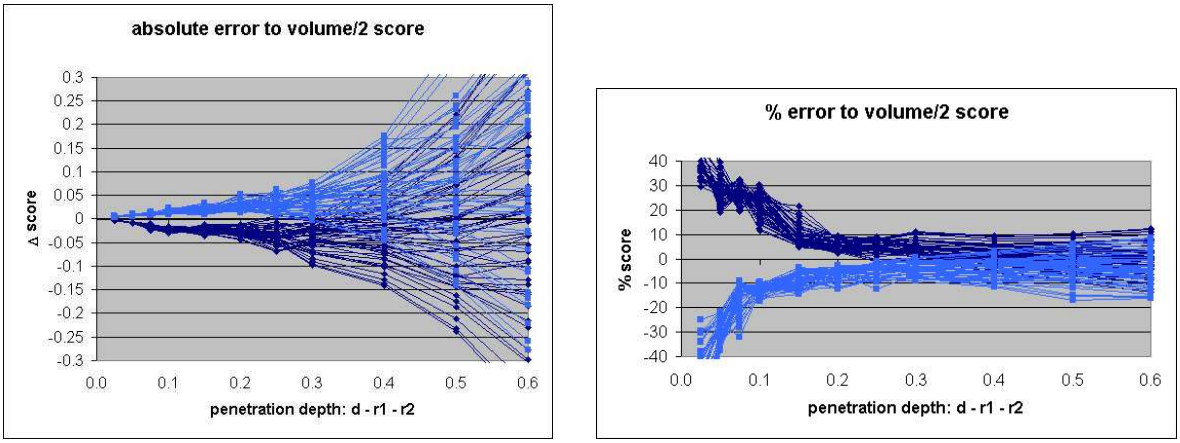


Figure 9: Absolute and relative volumes for all pairs of radii.