FEEDBACK ALGORITHM AND WEB-SERVER FOR PROTEIN STRUCTURE ALIGNMENT

Zhiyu Zhao*
Department of Computer Science, University of New Orleans,
New Orleans, LA 70148, USA
**Email: zzha2@cs.uno.edu*

Bin Fu
Department of Computer Science, University of Texas–Pan American,
Edinburg, TX 78539, USA
Email: binfu@cs.panam.edu

Francisco J. Alanis
Department of Computer Science, University of Texas–Pan American,
Edinburg, TX 78539, USA
Email: fjalanis@gmail.com

Christopher M. Summa
Department of Computer Science, University of New Orleans,
New Orleans, LA 70148, USA
Email: csumma@cs.uno.edu

We have developed a feedback algorithm for protein structure alignment between two protein backbones. A web portal implementing this method has been constructed and is freely available for use at http://fpsa.cs.uno.edu/ with a mirror site at http://fpsa.cs.panam.edu/FPSA/. We compare our algorithm with three other, commonly used methods: CE, DaliLite and SSM. The results show that in most cases our algorithm outputs a larger number of aligned positions when the (C_α) $RMSD$ is comparable. Also, in many cases where the number of aligned positions is larger or comparable, our learning method is able to achieve a smaller (C_α) $RMSD$ than the other methods tested. This trend of larger number of aligned positions and smaller (C_α) $RMSD$ is observed more frequently in cases where the similarity between protein structures is weak.

1. INTRODUCTION

Protein structure alignment attempts to compare the structural similarity between protein backbone chains. A protein molecule can have one or more protein chains, and each chain consists of a series of amino acid residues connected by peptide bonds. Protein structural similarity can be used to infer evolutionary relationships, or in classifying protein structures into more generalized groups. Typically, in protein structure comparison process, each protein chain is represented by an ordered set of 3-D points where each point corresponds to an alpha-carbon (C_α) atom in an amino acid residue. To compare the structural similarity between these "backbone" representations, a protein structure alignment algorithm seeks an optimal transformation by which chains are matched as closely as possible. An alignment is characterized by (1) how many positions are matched, (2) where these positions are, and (3) how well they are matched. (1) and (2) are available once an alignment is determined. For (3), a transformation based alignment algorithm usually calculates (C_α) $RMSD$, the root mean square distance between aligned positions.

The alignment problem is non-trivial – in fact, the problem of finding the optimal global alignment between protein structures has been shown to be NP-hard [12, 6]. Therefore, there have been a number of protein structure alignment algorithms presented in the past years (e.g. Refs. 1, 3, 4, 7, 8, 9, 11, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25), among

*Corresponding author.

them: DALI [7] (distance matrix based method), SSM [11] (secondary structure matching), CE [16] (the combinatorial extension method), and FATCAT [21] (protein structure alignment based on flexible transformation) are commonly used. We have developed a feedback algorithm for pairwise protein structure alignment and our web alignment tool is available for public access. Our algorithm is named SLIPSA, which stands for Self Learning and Improving Protein Structure Alignment. SLIPSA is self learning in that it has a feedback loop which sends the current alignment result back to its input in order to learn a better result in the next stage. In addition, SLIPSA accepts any reasonable upper-bound (C_α) $RMSD$ value as one of the inputs, and outputs an alignment result with an (C_α) $RMSD$ never greater than that value. Like CE, DALI and SSM, the SLIPSA alignment method is based on rigid body transformation, as opposed to flexible transformation-based algorithms such as the one described in Ref. 21.

Our paper is organized as follows: section 2 presents the SLIPSA algorithm; section 3 describes the framework and procedures used in SLIPSA; section 4 reports the experimental results of SLIPSA and compares it with some well known algorithms such as CE, DaliLite (the pairwise version of DALI), and SSM, each of them having a public website; section 4.3 discusses the results and concludes the paper.

2. SLIPSA: AN ALGORITHM WITH FEEDBACK

SLIPSA can be traced to a preliminary algorithm that we reported previously in Ref. 25, but the former has proceeded far beyond the latter in terms of maturity, stability, efficiency and availability. The SLIPSA algorithm first searches all the locally similar sub-chain pairs from two protein backbone chains. Such sub-chain pairs are called local alignments. Next, consistent local alignments are grouped into global alignment candidates called "double-center stars" and a currently optimal global alignment is chosen from all the candidates. Then this output is sent back to its own input in order to learn from itself. We call this a feedback. Such feedback is repeated to obtain improved results, until finally an optimal alignment (i.e. a result with as many as possible aligned C_α pairs and an acceptable (C_α) $RMSD$). SLIPSA can also learn from other algorithms when they are available.

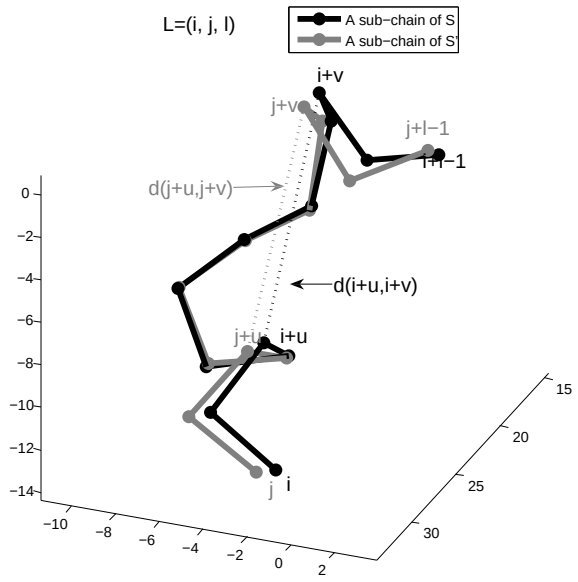

Fig. 1. Local alignment L=(i, j, l)

2.1. General Algorithmic Concepts

As shown in Figure 1, local alignments are discovered by checking the distance difference between corresponding C_α pairs. A local alignment $L = (i, j, l)$ is defined as the longest consecutive stretch of C_α pairs starting from position i in protein backbone chain $S = p_1 \cdots p_n$ and position j in backbone chain $S' = q_1 \cdots q_m$ and having length l, such that $|d(p_{i+u}, p_{i+v}) - d(q_{j+u}, q_{j+v})| \leq 2\epsilon$ for any $0 \leq u, v \leq l-1$ and $u \neq v$, where $d(p, q)$ is the Euclidean distance between two 3-D points p and q, and ϵ is a small constant. A local alignment has to be long enough to make sense.

After local alignments are discovered, they are organized into groups. Ideally, only consistent local alignments should be added to the same group. Suppose there are two local alignments $L_1 = (i_1, j_1, l_1)$ and $L_2 = (i_2, j_2, l_2)$, the point set $P = \{p_{i_1}, \cdots, p_{i_1+l_1-1}, p_{i_2}, \cdots, p_{i_2+l_2-1}\}$ is all the aligned points in the first chain, including those in L_1 and L_2, and $Q = \{q_{j_1}, \cdots, q_{j_1+l_1-1}, q_{j_2}, \cdots, q_{j_2+l_2-1}\}$ is all the aligned points in the second chain, also including

those in L_1 and L_2. We say that local alignments L_1 and L_2 are consistent if, after applying a rigid body transformation to Q, the (C_α) $RMSD$ between P and transformed Q is small enough. In other words, if we have a set of local alignments, we conclude that all these local alignments are consistent if all the local alignments share a common rigid body transformation which makes them consistent with each other. Therefore a global alignment can be defined as such a set of consistent local alignments with a common transformation and an acceptable (C_α) $RMSD$.

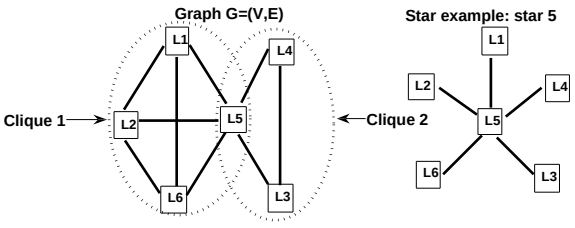

Fig. 2. An example star

The consistency relationship between local alignments can be represented as a graph. Given $A_L = \{L_1, L_2, \cdots, L_w\}$ where each $L_u = (i_u, j_u, l_u)$ $(1 \leq u \leq w)$ is a local alignment. A graph $G = (V, E)$ is defined accordingly, where each local alignment is a vertex of the graph, $V = A_L$ is the vertex set and E is the edge set. Edge $e_{uv}, e_{vu} \in E$ if and only if L_u and L_v are consistent. With this representation, grouping mutually consistent local alignments is equivalent to finding cliques in a graph, which is an NP-complete problem. A possible simplification to this problem is to look for "stars" rather than cliques in a graph. A star is a set of vertices including a center and all the other vertices that are connected to the center vertex. Since any clique must be included in some star, for our particular problem this simplification will not miss useful vertices. Figure 2 shows a graph, two cliques and an example star. There are 6 stars in the graph since $|V|$=6. They are $Star_1 = Star_2 = Star_6 = \{L_1, L_2, L_5, L_6\}$, $Star_3 = Star_4 = \{L_3, L_4, L_5\}$ and $Star_5 = \{L_1, L_2, L_3, L_4, L_5, L_6\}$. A set of all the unique stars is $Stars = \{Star_1, Star_3, Star_5\}$. Note that each star is finally a set of local alignments and each local alignment is a set of C_α pairs.

For each unique star, a corresponding global alignment candidate is calculated by deleting badly aligned C_α pairs involved in that star. Then all the candidates are compared and the optimal one is chosen. An example global alignment between protein chains 1ATP:E and 1PHK is shown in Figure 3, where N_{mat} is the number of aligned C_α pairs, (C_α) $RMSD$ is the root mean square distance between the aligned pairs, and the rigid body transformation used to align the two chains is T (the translation vector) and R (the rotation matrix).

The "star" approach has been used in a preliminary version of this algorithm [25], which has one center for each of its stars and shows some instability for aligning large proteins. We introduce the double-center method to group the local alignments and it is described in section 2.2. This greatly improves the reliability of the algorithm. Another crucial new technical development of this paper is the learning strategy based on feedback, which is described in section 2.3. The combination of two new methods greatly improves speed, reliability, and accuracy of the algorithm.

2.2. Introduction of Double-Center "Stars"

The single-center star method is not flawless. It works well when the two protein chains match well or the chain diameters are small. However, we have found that it is less stable when the chains do not match very well or the chain diameters are large. This is caused by deleting badly matched C_α pairs from each star, a method applied to obtain a global alignment candidate. When local alignments are grouped into an initial star, there may exist point pairs which do not match well. An initial transformation is calculated and the worst matched pair based on that transformation is first deleted, then the transformation is recalculated to select the second worst pair. This process is repeated. In this way the well matched pairs survive and the (C_α) $RMSD$ becomes smaller and smaller, until an acceptable (C_α) $RMSD$ is achieved. The effect of deleting bad point pairs relies on a good initial transformation, which in turn depends on the star center selection. With a single star center, the initial transformation has

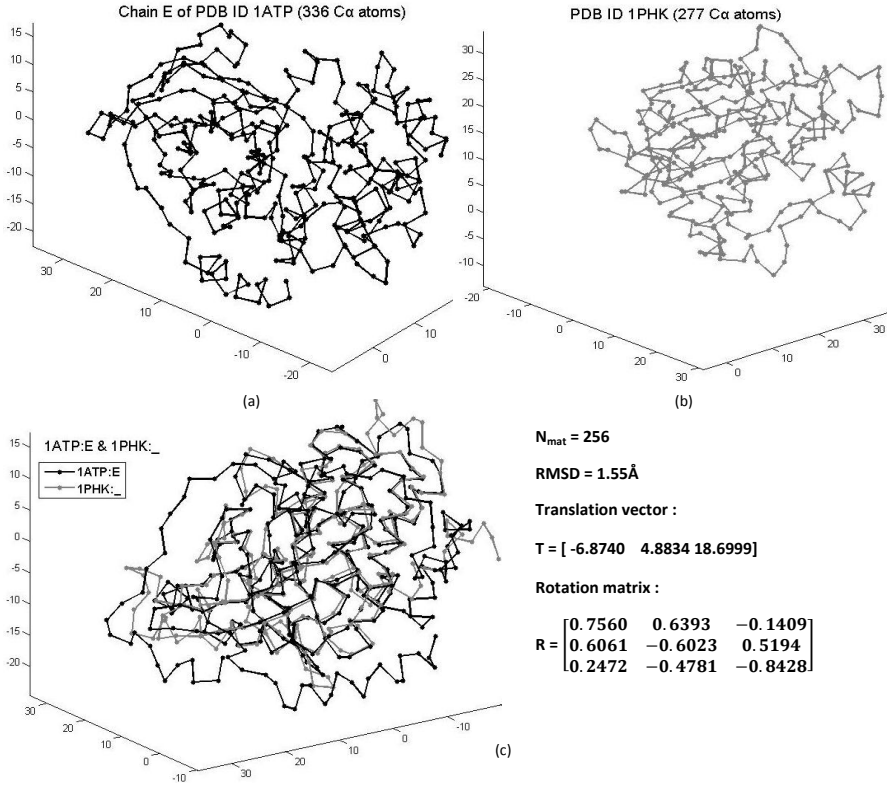

Fig. 3. A global alignment

great freedom to move and rotate in the point pair deletion process, thus the deletion may go along a more unpredictable way. This is more obvious when the local alignments are relatively short, which usually happens when the chains do not match very well or the chain diameters are large. Based on this observation, we consider grouping local alignments into double-center "stars".

A "double-center star" is, as suggested by its name, a "star" with two centers. Each single-center star can be extended to a corresponding double-center star, while the latter is much more stable. In a single-center star, each local alignment consistent with the center is added to the star, while in a double-center star, a local alignment can be added only when it is consistent with both centers. The first center of a double-center star is exactly the one in a single-center star, and the second center is selected from that star. The selection of the second center satisfies the following conditions: (1) it is consistent with the first center, (2) it is long enough to make sense, and (3) it is as far as possible from the first center. Figure 4 illustrates a double-center star corresponding to star 5 in Figure 2, suppose L_2 is the second center.

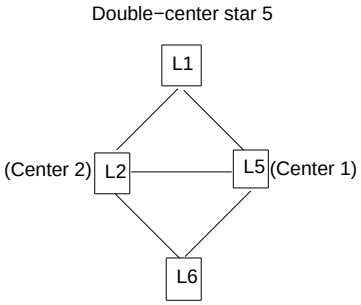

Fig. 4. A double-center star

Each local alignment in a single-center star is consistent with the center, however, this does not automatically guarantee that all the local alignments in the star are consistent. The consistency relationship is not necessarily transitive. To reduce the probability of adding inconsistent local alignments to the

star, a double-center star accepts local alignments in a more prudent way. It rejects the local alignments originally surviving in the single-center star on a weak basis, therefore local alignments in the double-center star are more likely to be those very good ones. To some extent, the presence of the second center has the effect of "extending" the local alignment in the first center. With such a long "local alignment" as the center, the star will be more stable because points in it have much less freedom to move or rotate. From another aspect, with this improvement the extent to which the initial transformation will change along with the deletion of bad point pairs is reduced significantly - the initial transformation will be closer to the final one, and thus the deletion will cause less unpredictability. Furthermore, the filtering of unpromising local alignments reduces their negative contribution to the overall transformation (as well as the number of point pairs involved in the initial star), speeding up the deletion process and resulting in a faster and better global alignment.

2.3. Development of Learning Ability

2.3.1. *Self-learning*

As we have mentioned, good star centers produce promising stars and have a greater probability of generating good global alignments. However, thus far the selection of star centers has been naïve: any local alignment with sufficient length can be the center of a star. The double-center method helps remove some unpromising local alignments from a star when the first center is determined, but it contributes nothing to the selection of the first center. If the first center of a star can be selected intelligently rather than by arbitrarily picking up a local alignment, then the star may yield a better global alignment. This intelligence may be difficult to achieve without any a priori knowledge on the global structural similarity between the two chains. However, when such knowledge is available, it is possible to improve the alignment by way of a self-learning strategy.

Once a currently optimal global alignment is output, we are able to know approximately where the aligned positions are. We organize the consecutively aligned point pairs into groups, and each group of consecutive point pairs is called a global alignment segment. A global alignment segment looks exactly like a local alignment, while as a part of a good global alignment, it should be a good "local alignment". Here local alignment is quoted because global alignment segments are not output of the local alignment phase, although there is no substantial difference between both definitions. To take advantage of these good global alignment segments, we apply a feedback mechanism to teach our alignment algorithm how to improve itself. The self-learning is implemented *via* the iterative utilization of its own output. When a global alignment is ready, consecutive alignment segments are extracted, then each segment is used as a new star center and local alignments consistent with the center are added to its group. This global alignment phase is repeated with a few new stars obtained from the currently optimal global alignment, until the alignment output converges (i.e. no changes are found between two iterations).

2.3.2. *Learning from others*

When a global alignment from another algorithm is available, the global alignment segments in that result can work as initial star centers. These centers are likely to be better than our own local alignments because they are from an optimal alignment result obtained from another algorithm. With these centers, our global alignment searching starts from a very good jumping-off point, therefore it is possible to output a result better than without learning. Learning from other algorithms may be more effective in the cases our algorithm performs worse than others. When it performs better than other algorithms even without learning, this learning may be less necessary, however it is never harmful, because if it results in a worse global alignment, its results can simply be disregarded. Therefore the combination of self-learning and learning-from-others will never output an alignment worse than the one of another algorithm. In the worst case it outputs nothing different. For this reason, our algorithm can also be used to improve the result of any other algorithm. We call this a refinement to that algorithm.

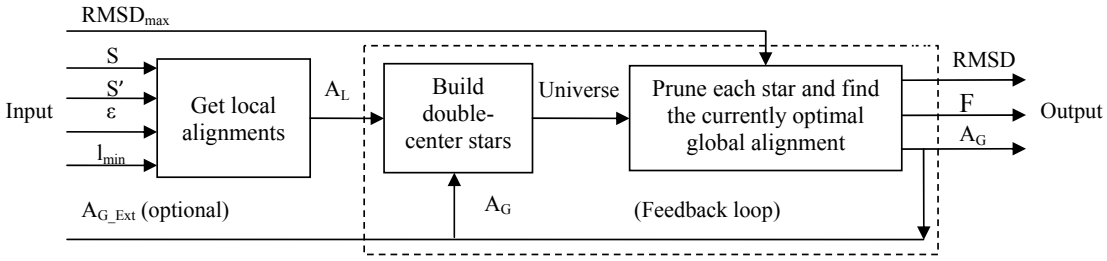

Fig. 5. The SLIPSA framework

3. THE FORMAL DESCRIPTION OF SLIPSA

We give the formal description of SLIPSA. Combining the double-center star, the self-learning and the learning-from-others methods which use feedback, we greatly improve our earlier work [25] and have found interesting results when comparing SLIPSA with some other algorithms. The SLIPSA framework is shown in Figure 5. This system takes six parameters: protein chains S and S', $RMSD_{max}$ (a user specified maximum (C_α) $RMSD$), distance constant ϵ, minimum local alignment length l_{min}, and an optional external global alignment A_{G_Ext}. Parameters S and S' are determined by the user, $RMSD_{max}$ is either determined by the user or obtained from another algorithm, ϵ and l_{min} are selected empirically according to the user input, and A_{G_Ext} is either empty or also obtained from the external algorithm. The system outputs an optimal global alignment result consisting of A_G (a set of global alignment segments), (C_α) $RMSD$ (a value not greater than $RMSD_{max}$) and F (a rigid body transformation corresponding to the final global alignment). The following sub-sections describe the details of the SLIPSA algorithm.

3.1. Getting Local Alignments

The calculation of local alignments has been reviewed in section 2.1. The procedure used to get local alignments can be from either Ref. 25 or other related papers (e.g. Ref. 21). The procedure body is omitted.

Get-Local-Alignments$(S, S', \epsilon, l_{\min})$
Input: protein backbone chains $S = p_1 \cdots p_n$, $S' = q_1 \cdots q_m$, distance constant ϵ and minimum local alignment length $l_{\min}$, where each p_i or q_i is a 3-D point corresponding to a C_α atom in a protein backbone.
Output: $A_L = \{L_1, L_2, \cdots, L_w\}$, a set containing all the local alignments of length $\geq l_{\min}$ between S and S'.

3.2. Building up Stars from Local Alignments

The improved procedure outputs double-center stars. The input is star centers from a set of global alignment segments, or from a local alignment set when the former is empty. The non-center nodes in a star are still chosen from the local alignment set.

Build-Double-Center-Stars(A_L, A_G)
Input: $A_L = \{L_1, L_2, \cdots, L_w\}$ and $A_G = \{L_{1'}, L_{2'}, \cdots, L_{w'}\}$, where A_L is a set of local alignments and A_G is a set of global alignment segments.
Output: $Universe = \{Star_1, Star_2, \cdots, Star_k\}$, a set of all the unique double-center stars.

begin
  $Universe \leftarrow \{\}$ (the empty set);
  if $(A_G = \{\})$ **then** $A \leftarrow A_L$;
   otherwise $A \leftarrow A_G$;
  for (each local alignment L_u in A)
    find $L_{u'}$, the second center based on L_u, in A;
    $Star_u \leftarrow \{L_u, L_{u'}\}$;
    for (each local alignment L_v in A_L)
      if (L_v is consistent with both L_u and $L_{u'}$) **then** $Star_u \leftarrow Star_u \cup \{L_v\}$;
    end for
    if $(Star_u \notin Universe)$

then $Universe \leftarrow Universe \cup \{Star_u\}$;
end for
return $Universe$;
end

3.3. Finding a Global Alignment from the Stars

In each iteration of our algorithm, a global alignment is output and used as an input of the next iteration. We describe how to prune the set of aligned pairs in a star and obtain the global alignment which has an (C_α) $RMSD$ not greater than that specified by the user. We refine a similar idea that was used in our original algorithm [25], which does not use feedback.

Prune-One-Star($Star, RMSD_{max}$)
Input: a $Star$ and $RMSD_{max}$ (a user specified maximum $RMSD$).
Output: $(A_S, RMSD_S, F_S, l_S)$, where $A_S = \{L_{1''}, L_{2''}, \cdots, L_{w''}\}$ is a set of global alignment segments which share a common transformation F_S with $RMSD_S \leq RMSD_{max}$, and l_S is the number of aligned point pairs in A_S.
begin
$A_S \leftarrow Star$;
$l_S \leftarrow$ the number of point pairs involved in A_S;
calculate transformation F_S and $RMSD_S$ for all the point pairs involved in A_S;
while $(RMSD_S > RMSD_{max})$
delete point pair (p, q) with the largest $d(p, F_S(q))$ in A_S;
$l_S \leftarrow l_S - 1$;
recalculate transformation F_S and $RMSD_S$ for all the point pairs involved in A_S;
end while
return $(A_S, RMSD_S, F_S, l_S)$;
end

In the following function Find-Global-Alignment(), we apply the Prune-One-Star() procedure to each of the stars in the universe which is built from Build-Double-Center-Stars(). The alignment that contains the largest number of aligned pairs will be returned.

Find-Global-Alignment($Universe, RMSD_{max}$)
Input: $Universe = \{Star_1, Star_2, \cdots, Star_k\}$ and $RMSD_{max}$ (a user specified maximum $RMSD$).
Output: $(A_G, RMSD, F)$, where $A_G = \{L_{1'}, L_{2'}, \cdots, L_{w'}\}$ is a set of global alignment segments which share a common transformation F with $RMSD \leq RMSD_{max}$.
begin
sort $Universe$ by a descending order of the number of 3-D point pairs involved in each star;
$l_{max} \leftarrow 0$;
for (each $Star_u$ in $Universe$)
$(A_S, RMSD_S, F_S, l_S) \leftarrow$ Prune-One-Star $(star_u, RMSD_{max})$;
if $(l_S > l_{max})$ **then** $A_G \leftarrow A_S$; $RMSD \leftarrow RMSD_S$; $F \leftarrow F_S$; $l_{max} \leftarrow l_S$;
end for
return $(A_G, RMSD, F)$;
end

3.4. The Feedback Procedure

This is the main procedure of SLIPSA. It calls Get-Local-Alignments in the first step, then Build-Double-Center-Stars and Find-Global-Alignment are called repeatedly. A global alignment output by the current iteration serves as the input of the next iteration. The procedure terminates when the global alignment ceases to change (i.e. converges).

SLIPSA($S, S', \epsilon, l_{min}, RMSD_{max}, A_{G\_Ext}$)
Input: S, S', ϵ, l_{min}, $RMSD_{max}$ and A_{G_Ext}, where A_{G_Ext} can be either empty or a set of global alignment segments obtained from an external algorithm.
Output: $(A_G, RMSD, F)$.
begin
$A_L \leftarrow$ Get-Local-Alignments$(S, S', \epsilon, l_{min})$;
$A_G \leftarrow A_{G\_Ext}$;
do
$A'_G \leftarrow A_G$;
$Universe \leftarrow$ Build-Double-Center-Stars (A_L, A'_G);
$(A_G, RMSD, F) \leftarrow$ Find-Global-Alignment $(Universe, RMSD_{max})$;
while $(A_G \neq A'_G)$;
return $(A_G, RMSD, F)$;
end

When no external alignment is available, procedure SLIPSA is called by way of SLIPSA(S, S', ϵ, l_{min}, $RMSD_{max}$, {}). When it is available, SLIPSA can be called as SLIPSA($S, S', \epsilon, l_{min}, RMSD_{max}, A_{G\_Ext}$). We call this a refinement to external alignment A_{G_Ext}. To independently test the performance of our algorithm,

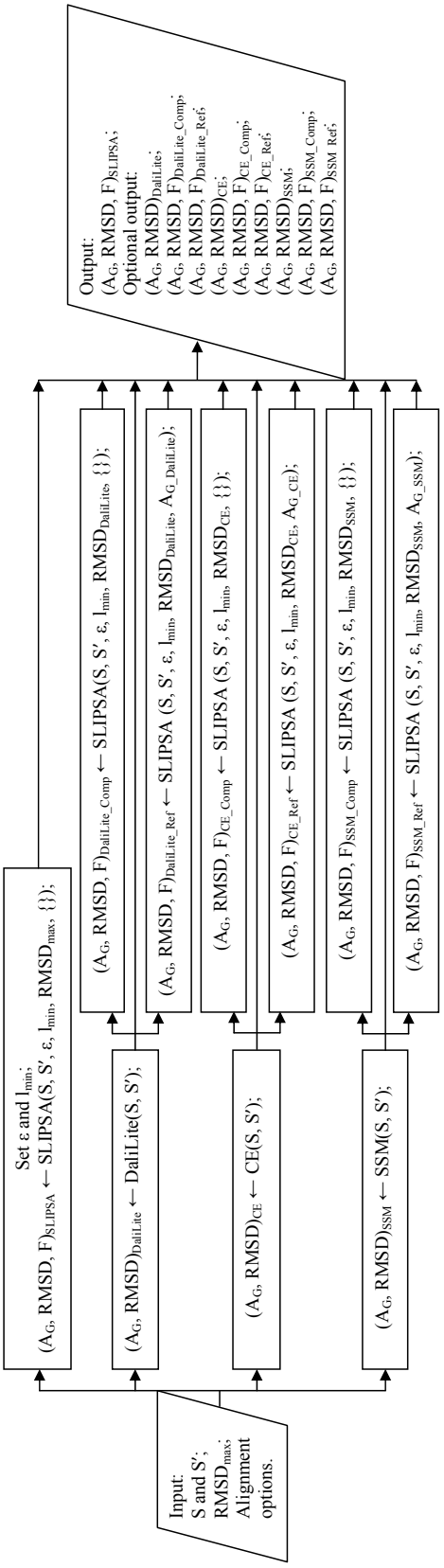

Fig. 6. The web alignment work flow

none of the experiments reported in section 4 uses any external alignment as our input.

4. EXPERIMENTAL ENVIRONMENT AND RESULTS

4.1. Our Web Alignment Tool

We have developed a web alignment tool based on the SLIPSA algorithm. The website is available for public access at http://fpsa.cs.uno.edu/ with a mirror site at http://fpsa.cs.panam.edu/FPSA/. It is not only a SLIPSA alignment tool but also an alignment comparison tool between SLIPSA and DaliLite, CE and SSM, some commonly used protein structure alignment algorithms with public websites.

The data used for protein alignment are the PDB files downloaded from the RCSB Protein Data Bank. The files have been moved to the Worldwide Protein Data Bank (wwPDB) by the time we wrote this paper. As of March 2008, there were over 49,000 protein structures with over 100,000 chains discovered.

Our website is built on an Intel dual-Xeon 3G Hz PC server with 3GB memory. The web development tools we have used include Apache HTTP server with PHP support, ActivePerl and MySQL database server. The SLIPSA algorithm is written in MATLAB. See Refs. 20 and 2 for the rigid body transformation method that we have used in SLIPSA.

The work flow of our website is shown in Figure 6. Besides a maximum value for (C_α) $RMSD$, it accepts either PDB IDs or user uploaded PDB files as input. It is optional to compare SLIPSA with DaliLite, CE or SSM. When a comparing option is chosen, our tool automatically submits alignment request to and retrieves result from DaliLite, CE or SSM website, and performs SLIPSA alignment according to the retrieved (C_α) $RMSD$ value. The website outputs the following alignment results. Beyond the first result listed, all others are optional depending on the user choices. Note that SLIPSA outputs A_G (a set of global alignment segments), (C_α) $RMSD$ and F (a rigid body transformation).

(1) $(A_G, RMSD, F)_{\text{SLIPSA}}$: the SLIPSA result with a user specified $RMSD_{max}$.
(2) $(A_G, RMSD)_{DaliLite}$: the DaliLite result retrieved automatically from its website.
(3) $(A_G, RMSD, F)_{DaliLite\_Comp}$: the SLIPSA result with an $RMSD$ retrieved from DaliLite website as input. This result is used to compare SLIPSA with DaliLite.
(4) $(A_G, RMSD)_{CE}$: the CE result retrieved automatically from its website.
(5) $(A_G, RMSD, F)_{CE\_Comp}$: the result used to compare SLIPSA with CE.
(6) $(A_G, RMSD)_{SSM}$: the SSM result retrieved automatically from its website.
(7) $(A_G, RMSD, F)_{SSM\_Comp}$: the result used to compare SLIPSA with SSM.

4.2. Experimental Results

We have collected 224 alignment cases to test the performance of our algorithm. The test cases were originally proposed by various papers for various testing purposes. They include No. 1 - No. 20 (see Table III in Ref. 16), No. 21 - No. 88 (see Table I in Ref. 5), No. 89 (see Tables I and II in Ref. 16), No. 90 - No. 92 (supplement to Table III in Ref. 16), No. 93 (see Figure 5 in Ref. 16), No. 94 - No. 101 (see Table IV in Ref. 16), No. 102 - No. 111 (see Table V in Ref. 16), No. 112 - No. 120 (supplement to Table V in Ref. 16), No. 121 - No. 124 (see Table VII in Ref. 16), No. 125 - No. 143 (see Table 1 in Ref. 15), No. 144 - No. 183 (see Table 1 in Ref. 22) and No. 184 - No. 224 (see Table 2 in Ref. 22). Due to the space limit, the PDB IDs of those proteins are not listed in this paper and they can be provided upon request.

Based on this test set, we compare SLIPSA with DaliLite, CE and SSM in terms of N_{mat} (the number of aligned positions) and (C_α) $RMSD$. Common protein alignment scoring methods such as Z-score, Q-score, P-score and geometric measures proposed in Ref. 10 all take N_{mat} and (C_α) $RMSD$ into account. Because of the $RMSD$ flexibility of SLIPSA, it is easy to compare SLIPSA with DaliLite, CE and SSM on the basis of N_{mat} because in most cases SLIPSA outputs an equal (C_α) $RMSD$. In each test case SLIPSA outputs an (C_α) $RMSD$ not greater than that of DaliLite, CE, or SSM. If N_{mat} of SLIPSA is larger than N_{mat} of DaLiLite, CE, or SSM, we call it an N_{mat} increment. Similarly, if the (C_α) $RMSD$ of SLIPSA is smaller than the (C_α) $RMSD$ of DaLiLite, CE or SSM, we call it a

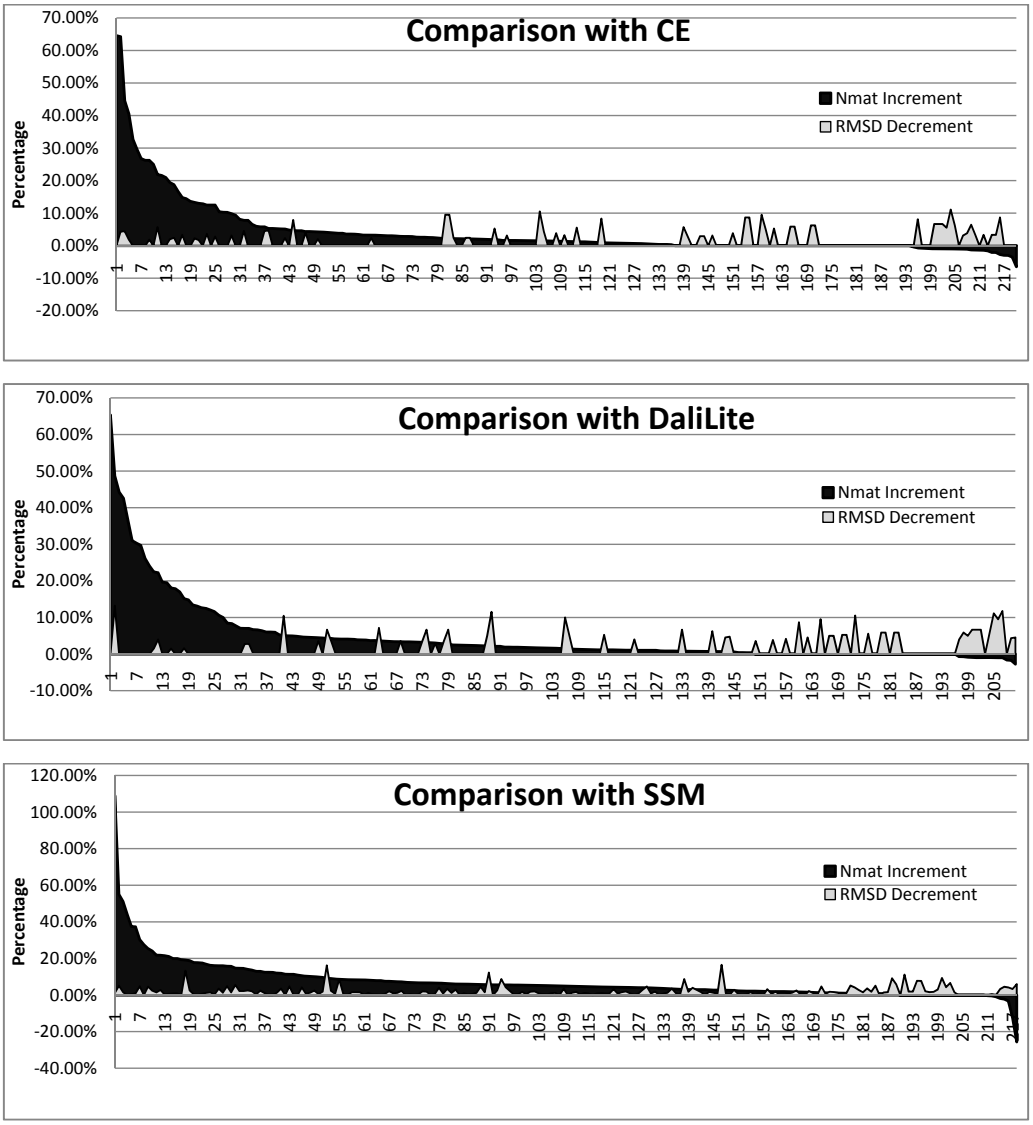

Fig. 7. Comparing SLIPSA with DaliLite, CE and SSM

(C_α) $RMSD$ decrement. The N_{mat} increment rate is calculated by ($N_{mat\_\text{SLIPSA}} - N_{mat\_X}$) / N_{mat_X} and the (C_α) $RMSD$ decrement rate is calculated by ($RMSD_X - RMSD_{\text{SLIPSA}}$) / $RMSD_X$, where X is DaliLite, CE or SSM. Figure 7 illustrates such increments and decrements in percentage. For the convenience of illustration, the results are sorted in a descending order of the N_{mat} increment rate. Due to the space limit, the detailed result data are not listed in this paper. They can be provided upon request. It should be mentioned that, (1) no SSM comparison was performed in our earlier paper [25]; (2) in a few cases that we could not find results on the DaliLite, CE or SSM websites, we marked the cases as "n/a" and did not compare SLIPSA with them; (3) from the time we completed this paper, it is possible to see result changes on any of the alignment websites and we have observed minor changes on some of them; (4) the SLIPSA experiments did not use any external alignment as input, although our algorithm is able to refine the alignment results retrieved from other web servers.

4.3. Discussion on the Results

Table 1 shows some statistical data based on the results in Figure 7. For each case in which an

Table 1. Statistics on the experimental results

	DaliLite	CE	SSM
Number of valid cases	210	220	218
Cases with larger N_{mat} by SLIPSA	149(66.67%)	136(61.82%)	189(86.70%)
Cases with smaller N_{mat} by SLIPSA	14(6.67%)	26(11.82%)	8(3.67%)
Maximum N_{mat} increment by SLIPSA	49	56	51
Maximum N_{mat} decrement by SLIPSA	2	9	12
Maximum N_{mat} increment rate by SLIPSA	65.33%	64.58%	109.09%
Maximum N_{mat} decrement rate by SLIPSA	2.74%	6.45%	25.53%
Average N_{mat} increment by SLIPSA	4.15	3.63	7.24
Average N_{mat} increment rate by SLIPSA	4.56%	4.13%	7.37%
Cases with smaller $RMSD$ by SLIPSA	56(26.67%)	64(29.09%)	177(81.19%)
Maximum $RMSD$ decrement by SLIPSA	0.7	0.4	0.52
Maximum $RMSD$ decrement rate by SLIPSA	13.21%	11.11%	16.56%
Average $RMSD$ decrement by SLIPSA	0.04	0.04	0.05
Average $RMSD$ decrement rate by SLIPSA	1.55%	1.42%	2.07%

Table 2. Comparison based on weak similarity

	DaliLite		CE		SSM	
	Valid Cases	Avg. N_{mat} Inc.	Valid Cases	Avg. N_{mat} Inc.	Valid Cases	Avg. N_{mat} Inc.
$RMSD \geq 5.0$	12	26.48%	14	21.62%	0	/
$RMSD \geq 4.0$	20	23.48%	41	14.75%	9	17.50%
$RMSD \geq 3.0$	77	10.09%	102	7.64%	51	12.15%

alignment result was missing from either DaliLite, CE or SSM, we did not compare it with SLIPSA. Also, since DaliLite, CE and SSM may have different (C_α) $RMSD$ values for a given test case, they were not compared mutually. In our experiments, when compared with DaliLite, CE and SSM respectively, SLIPSA outputs a larger N_{mat} in 66.67%, 61.82% and 86.70% of the cases; The maximum N_{mat} increment rate of SLIPSA is 65.33%, 64.58% and 109.09%; Averagely, SLIPSA increases 4.56%, 4.13%, and 7.37% of the N_{mat}; In 26.67%, 29.09% and 81.19% of the cases SLIPSA outputs a smaller (C_α) $RMSD$ with the maximum (C_α) $RMSD$ decrement rate being 13.21%, 11.11% and 16.56%. To sum up, in most cases we see SLIPSA results with a larger or same N_{mat} and a same or smaller (C_α) $RMSD$. In some cases that SLIPSA outputs a smaller N_{mat}, we also see a smaller (C_α) $RMSD$.

We also attempt to compare SLIPSA with DaliLite, CE and SSM in the cases of weak similarities. To simplify the comparison process, we tentatively define a weak similarity as a large (C_α) $RMSD$ between aligned chains. This definition may be incomplete, however, we have already observed some interesting results. For example, when compared with DaliLite and CE, the average N_{mat} increment rates of SLIPSA are 4.56% and 4.13% respectively, while in the cases with (C_α) $RMSD \geq 5.0$, the numbers are 26.48% and 21.62%, much higher than the overall average values. See Table 2 for details. In brief, SLIPSA obtains high average N_{mat} increment rate in weak similarity cases, and the larger the (C_α) $RMSD$, the higher the average N_{mat} increment rate.

The running time of each algorithm was recorded. The average running time of DaliLite, CE and SSM is 16.86s, 6.14s and 9.15s, respectively. When compared with them (i.e. using the $RMSD$ from the best fit from the comparison algorithms as the $RMSD$ upper-bound in SLIPSA), the average running time of SLIPSA is 105.97s, 69.89s and 81.43s, respectively. In about 50% of the cases the SLIPSA average time is below the DaliLite average, and the corresponding numbers for CE and SSM are about 25% and 28%, respectively. Possible ways to reduce the running time are discussed below.

(1) The web server was built on a slow machine. We have also tested the algorithm on an IBM ThinkPad laptop computer with Core2 Duo

1.8GHz CPUs. This machine was much slower than the mainstream web server machines, while the same results took only $\frac{1}{2}$ to $\frac{2}{3}$ of the time used on our current web server. It is possible to improve the speed to great extent by using a machine with high computational performance. (2) We used Matlab to implement the algorithm. Matlab facilitates the proof-of-concept development of complicated scientific programs, however, according to our experience it is possible to speed up algorithms at least several times if they are implemented in other languages such as C, C++ and Java. In addition, parallel and distributed programming on high performance computational resources can also help reduce the execution time. (3) The algorithm is slower when the proteins are long and/or the (C_α) $RMSD$ is large. In such cases the number of local alignments are large and the graph complexity is high. However, the algorithm can be optimized to reduce the complexity. Possible methods include reducing the dimension of data, removing unpromising local alignments as early as possible, limiting the number of times of feedback, and so on.

References

1. Chew LP, Huttenlocher D, Kedem K, Kleinberg J. Fast detection of common geometric substructure in proteins. *Journal of Computational Biology* 1999; **6(3-4)**: 313–325.
2. Lorusso A, Eggert DW, Fisher RB. A comparison of four algorithms for estimating 3-D rigid transformations. *British Machine Vision Conference* 1995; 237–246.
3. Falicov A, Cohen FE. A surface of minimum area metric for the structureal comparison of proteins. *Journal of Molecular Biology* 1996; **258**: 871–892.
4. Fischer D, Nussinov R, Wolfson H. 3D substructure matching in protein molecules. *Proc. 3rd Intl Symp. Combinatorial Pattern Matching, LNCS* 1992; **644**: 136–150.
5. Fischer D, Elofsson A, Rice D, Eisenberg D. Assessing the performance of fold recognition methods by means of a comprehensive benchmark. *Proc. 1st Pacific Symposium on Biocomputing* 1996; 300–318.
6. Godzik A. The structural alignment between two proteins: Is there a unique answer? *Protein Science* 1996; **5**: 1325–1338.
7. Holm L, Sander C. Protein structure comparison by alignment of distance matrices. *Journal of Molecular Biology* 1993; **233**: 123–138.
8. Ilyin VA, Abyzov A, Leslin CM. Structural alignment of proteins by a novel TOPOFIT method, as a superimposition of common volumes at a topomax point. *Protein Science* 2004; **13**: 1865–1874.
9. Kolodny R, Linial N, Levitt M. Approximate protein structural alignment in polynomial time. *Proc. Natl. Acad. Sci. USA* 2004; **101(33)**: 12201-12206.
10. Kolodny R, Koehl P, Levitt M. Comprehensive evaluation of protein structure alignment methods: scoring by geometric measures. *Journal of Molecular Biology* 2005; **346(4)**: 1173-1188.
11. Krissinel E, Henrick K. Secondary-structure matching (SSM), a new tool for fast protein structure alignment in three dimensions. *Acta Cryst.* 2004; **D60**: 2256–2268.
12. Lathrop RH. The protein threading problem with sequence amino acid interaction preferences is NP-complete. *Protein Engineering* 1994; **7**: 1059–1068.
13. Lessel U, Schomburg D. Similarities between protein 3-D structures. *Protein Engineering* 1994; **7(10)**: 1175–87.
14. Madej T, Gibrat JF, Bryant SH. Threading a database of protein cores. *Proteins* 1995; **23**: 356–369.
15. Ortiz AR, Strauss CEM, Olmea O. MAMMOTH (matching molecular models obtained from theory): an automated method for model comparison. *Protein Science* 2002; **11**: 2606–2021.
16. Shindyalov IN, Bourne PE. Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Engineering* 1998; **11**: 739–747.
17. Singh AP, Brutlag DL. Hierarchical protein superposition using both secondary structure and atomic representation. *Proc. Intelligent Systems for Molecular Biology* 1997; 284–293.
18. Taylor WR, Orengo CA. Protein structure alignment. *Journal of Molecular Biology* 1989; **208(1)**: 1–22.
19. Taylor WR. Protein structure comparison using iterated double dynamic programming. *Protein Science* 1999; **9**:654–665.
20. Umeyama S. Least-squares estimation of transformation parameters between two point patterns. *IEEE Tran. on Pattern Analysis and Machine Intelligence* 1991; **13(4)**: 376–380.
21. Ye Y, Godzik A. Database searching by flexible protein structure alignment. *Protein Science* 2004; **13(7)**: 1841–1850.
22. Ye J, Janardan R, Liu S. Pairwise protein structure alignment based on an orientation-independent backbone representation. *Journal of Bioinformatincs and Computational Biology* 2005; **4(2)**: 699–717.
23. Yona G, Kedem K. The URMS-RMS hybrid algorithm for fast and sensitive local protein structure alignment. *Journal of Computational Biology* 2005; **12**:12–32.
24. Zhang Y, Skolnick J. TM-align: a protein structure alignment algorithm based on the TM-score. *Nucleic Acids Research* 2005; **33**: 2302–2309.
25. Zhao ZY, Fu B. A Flexible algorithm for pairwise protein structure alignment. *the 2007 International Conference on Bioinformatics and Computational Biology* 2007; 16–22.