Protein Structure Alignment in Subquadratic Time

Aleksandar Poleksic

Department of Computer Science, University of Northern Iowa
Cedar Falls, Iowa, USA
poleksic@cs.uni.edu

Abstract. The problem of finding an optimal structural alignment for a pair of superimposed proteins is often amenable to the Smith-Waterman dynamic programming algorithm, which runs in time proportional to the product of the lengths of sequences being aligned. While the quadratic running time is acceptable for computing a single alignment of two, spatially "fixed", structures, the time complexity becomes a bottleneck when running the Smith-Waterman routine multiple times in order to find an optimal pairwise superposition. We present a subquadratic running time algorithm capable of computing an alignment that optimizes one of the most widely used measures of protein structure similarity, defined as the number of pairs of residues in two proteins that can be superimposed under a predefined distance cutoff. The algorithm presented in this article can be used to significantly improve the speed-accuracy tradeoff in a number of popular protein structure alignment methods.

Keywords: protein structure, structure comparison, alignment, dynamic programming.

1 Introduction

Automated methods for protein structure comparison are of critical importance in several fields, including protein three-dimensional structure prediction [1-5], functional site comparison [6,7,8], and protein structural and functional annotation [9,10,11]. Protein structure comparison problem is much more difficult than its closely related sequence alignment problem [12-15]. For methods that minimize the inter-atomic distances, such as STRUCTAL [16,17], TM-align [18], Fr-TM-align [19], CAALIGN [20], LOCK [21], or LGA [22], the sequence alignment problem can be viewed as a subproblem of the structure comparison problem, since the goal of the latter is to simultaneously find both, a superposition and an alignment that maximizes a given structure similarity measure. In fact, a common approach to finding an optimal structural superposition of two proteins is to solve multiple pairwise alignment problems, one for each inspected spatial superposition of the input protein structures.

One of the most intuitive and most widely used measures of pairwise structure similarity is the number of atoms in two proteins that can be superimposed under a specified distance cutoff. For now, we will denote this metric by $CA \leq d$, where d

J. Suzuki and T. Nakano (Eds.): BIONETICS 2010, LNICST 87, pp. 363–374, 2012.

denotes the distance threshold in Ångströms (and *CA* indicates that the structure of each protein is represented by its sequence of α-carbon atoms).

Many widely used protein structure similarity metrics build upon $CA \leq d$, including GDT_TS [22], MaxSub [23], AL0 [24], "*CA*-atoms < 3Å" [25,26] and Q-score [25]. GDT_TS is the main measure used in the CASP benchmark of methods for protein structure modeling [1]. This measure is defined as the average value of $\mathrm{GDT\_P_i}$, $i \in \{1, 2, 4, 8\}$, where $\mathrm{GDT\_P_i}$ represents the percentage of C_α atoms that can be superimposed under i Ångströms of the aligned atoms in the experimental structure. In the CAFASP experiment [27], the quality of a protein model is given by the model's MaxSub score, which represents the weighted fraction of the number of atoms in the model structure that can be fit under 3.5Å. The LiveBench experiment [28] uses "*CA*-atoms < 3Å" (in our notation $CA < 3$) and Q-score, among other metrics, to evaluate the sensitivity and the specificity of protein structure prediction servers. The Q-score measure is defined as "*CA*-atoms < 3Å" divided by the length of the model.

The widespread use of $CA \leq d$ establishes the need for an efficient algorithm for its optimization. For a pair of superimposed proteins, p and q, $CA \leq d$ can be maximized using a simplified version of the standard Smith-Waterman dynamic programming algorithm [29] with zero gap penalties. This algorithm first computes the score matrix

$$S(i,j) = \begin{cases} 1 & \text{if } \| p_i - q_j \| \leq d \\ 0 & \text{otherwise} \end{cases}$$

(where $\| p_i - q_j \|$ denotes the Euclidean distance between the p_i and q_j) and then fills out the dynamic programming matrix in order to find an optimal alignment of p and q. The cost of both procedures, i.e., the procedure for computing the score matrix and the procedure for filling out the dynamic programming matrix is $O(mn)$, where m and n denote the lengths of proteins p and q, respectively.

While the Smith-Waterman method is fast enough for computing a single alignment of two, fixed in space, proteins, the time complexity becomes a bottleneck when finding an optimal pairwise structural superposition by repeatedly running the Smith-Waterman procedure, once for each inspected spatial orientation of the input structures. To circumvent high computational cost, current methods for protein structure matching trade sensitivity for speed by utilizing heuristic techniques in search for a reasonable, suboptimal solution.

Here we present an $O(mn^{3/4})$ worst-case running time algorithm, guaranteed to maximize $CA \leq d$ for any pair of protein structures. Our benchmarking results show that, in typical protein structure matching applications, the speedup factor of our algorithm over the Smith-Waterman algorithm exceeds an order of magnitude. Hence, our algorithm can be readily applied to improve the tradeoff between the speed and the accuracy in a number of existing protein structure comparison methods, including some of the methods discussed above.

We emphasize that subquadratic alignment algorithms are only known for some special sequence alignment problems, such as the Longest Common Subsequence problem (LCS). Using the so-called "Four Russians Speedup" technique [30,31], LCS problem can be solved in $O(n^2/\log n)$ time.

2 Methods and Results

A protein p of length m can be viewed as a sequence of points in the three-dimensional space:

$$p = (p_1, \ldots, p_m), \quad p_i \in R^3, \text{ for } i \in \{1, \ldots, m\}.$$

In many applications, the p_i's represent the protein's C_α atoms.

An alignment of proteins $p = (p_1, \ldots, p_m)$ and $q = (q_1, \ldots, q_n)$ is a sequence of pairs of points from p and q:

$$A(p,q) = ((p_{i_1}, q_{i_1}), \ldots, (p_{i_k}, q_{i_k})),$$

where $1 \le i_1 \le \ldots \le i_k \le m$ and $1 \le j_1 \le \ldots \le j_k \le n$. We will use $A_d(p,q)$ to denote an alignment of p and q that maximizes $CA \le d$, i.e. the number of aligned pairs (p_i, q_j) at distance $\le d$.

The subquadratic running time algorithm, presented below, consists of two procedures: a procedure for computing the score matrix and a procedure for computing an optimal alignment. The total cost of our method is dominated by the cost of computing the score matrix, since our alignment routine runs on the order of $O(m \log n)$.

2.1 Computing the Score Matrix

Our algorithm first computes a "trim-down" version of the standard score matrix $S = S(i, j)$. More precisely, the algorithm, presented here, generates, for every point p_i from the protein p, a list $L(i) = (j_1, \ldots, j_l)$, $j_1 < \ldots < j_l$, of positions of all points from the protein q that are at distance $\le d$ from p_i. It should be noted that the length of $L(i)$ cannot exceed K_d, where K_d represents an upper bound on the number of C_α atoms that can be packed inside a sphere of radius d in R^3. This implies that the space requirement for storing the collection of all lists $L = (L(1), \ldots, L(m))$ (one for each point p_i from p) does not exceed $K_d \cdot m = O(m)$.

The most straightforward way of computing $L(i)$ is to calculate the distances $\| p_i - q_j \|$ between p_i and each q_j and then append j to the end of the list $L(i)$ if $\| p_i - q_j \| \le d$. The problem with this approach is that it requires $O(n)$ operations for

each p_i, resulting in $O(mn)$ total cost of the score matrix computation. To speed up the computation of $L(i)$, we first note that many distance calculations can be skipped due to the spacing of the protein's consecutive C_α atoms. Let $w > 0$ be the smallest integer such that $wd > c$, where c is an upper bound on the distance between two consecutive C_α atoms (c ~ 3.8Å) and let $k = \lceil wd \rceil$. If $\| p_i - q_j \| > 2k$ then $j+1$ does not belong to $L(i)$, since

$$\| p_i - q_{j+1} \| > \| p_i - q_j \| - \| q_{j+1} - q_j \| > 2wd - c > wd \geq d .$$

In general, if $\| p_i - q_j \| > (t+1)k$, where $t > 0$ is an integer, then none of $j+1, \ldots, j+t$ belongs to $L(i)$, rendering the calculations of distances between p_i and each $q_{j+1}, \ldots, q_{j+t}$ unnecessary.

Assuming the cubic lattice model of protein structures, we now prove that each list $L(i)$ can be computed in $O(n^{3/4})$ time.

Let B_t denotes the closed ball of radius $(t+1)k$ centered at p_i, where $t \geq 0$ is an integer (Fig. 1).

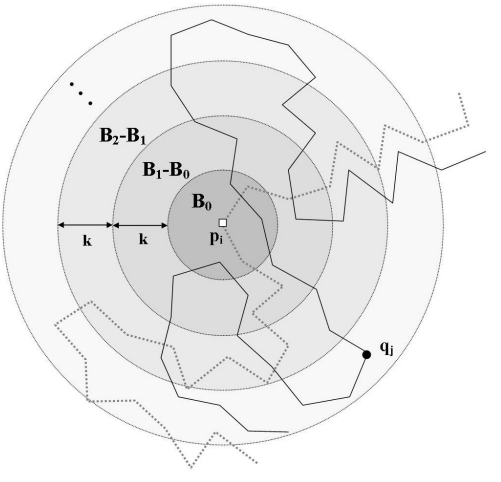


Fig. 1. A toy example of two protein structures, *p* and *q*, represented by dotted gray and black lines, respectively. If $q_j \in B_{t+1} - B_t$ then $S(i, j+l) = 0$ for every $l \in \{1,\ldots,t\}$.

For every inspected point q_j from the spherical shell $B_{t+1} - B_t$, at least *t* points from the protein *q* can be skipped, because $\| p_i - q_j \| > (t+1)k$. Because we are interested in an upper bound on the algorithm's cost, we can assume that the visited points from the protein *q* are packed as tightly as possible around the point p_i (this scenario results in the least number of skipped points *s* from *q*). The key observation here is that the total number of inspected points from *q* can be represented as

$v = b_0 + b_1 + b_2 + ... + b_h + a$, where b_t denotes the number of points from q that are packed inside $B_t - B_{t-1}$ (by definition B_{-1} is empty) and $0 \le a < b_{h+1}$. The total number of skipped points from q is $s \ge 1b_2 + 2b_3 + ... + (h-1)b_h$. According to the result of Chamizo and Iwaniec, $v = O(h^3)$ and $s = \Omega(h^4)$ (see Theorem 1.1, [3]) and, therefore, $v = O(s^{3/4})$. Since $s < n$, it follows that $v = O(n^{3/4})$.

The algorithm for computing the list *L* can be written as follows:

```
Algorithm SCORE_MATRIX
// Given the proteins p and q and the distance cutoff
// d, compute the score matrix L.
1.     for i←1 to m do
2.          j←1
3.          pos←1
4.          while j≤n do
5.               distance←‖p_i − q_j‖
6.               if  distance   ≤  d
7.                    L[i,pos]←j
8.                    j←j+1
9.                    pos←pos+1
10.              else
11.                   j←⌊(distance-d)/c⌋+j+1
```

It should be emphasized that the algorithm SCORE_MATRIX, is even more efficient than the general procedure we have just described, since it uses $\| p_i - q_j \| > d + tc$ as the criteria for skipping t points from q. While both $\| p_i - q_j \| > d + tc$ and $\| p_i - q_j \| > (t+1)k$ are sufficient conditions for skipping $q_{j+1}, ..., q_{j+t}$, the former results in a more efficient algorithm while the latter makes our proof easier to follow.

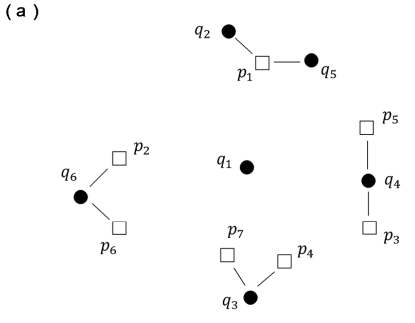
2.2 Computing Optimal Alignment

In this section we present $O(m \log n)$ algorithm for computing an optimal alignment $A_d(p,q)$ of p and q. In contrast to the method described below, a standard $O(mn)$ dynamic programming algorithm for $A_d(p,q)$ implements the following recurrence relation to compute the score $C(i,j)$ of an optimal alignment of the sub-structures $p^i = (p_1, ..., p_i)$ and $q^j = (q_1, ..., q_j)$:

$$C(i,j)=\begin{cases}0 & \text{if } i=0 \text{ or } j=0\\ C(i-1,j-1)+1 & \text{if } \| p_i-q_j \| \le d\\ \max\{C(i-1,j),C(i,j-1)\} & \text{otherwise}\end{cases}$$

As we will demonstrate shortly, the special binary form of the score matrix (Fig. 2) makes the protein structure alignment problem amenable to a much more efficient technique, a technique similar to one used for computing the longest common substring (LCS) of two strings over a finite alphabet [33,34]. It is interesting to note that our method has better (worst-case) running time than the corresponding $O(mn\log n)$ algorithm for LCS [33], due to a "sparse" score matrix for any given pair of protein structures. In order to describe the algorithm in more details, we first need some terminology.

We call a pair of indices (i,j) a *match* if $S(i,j)=1$ (i.e. if $\| p_i-q_j \| \le d$). It is not difficult to see that the collection M of all matches can be partitioned as



(b)

	q_1	q_2	q_3	q_4	q_5	q_6
p_1	0	1	0	0	1	0
p_2	0	0	0	0	0	1
p_3	0	0	0	1	0	0
p_4	0	0	1	0	0	0
p_5	0	0	0	1	0	0
p_6	0	0	0	0	0	1
p_7	0	0	1	0	0	0

(c)

		q_1	q_2	q_3	q_4	q_5	q_6
	0	0	0	0	0	0	0
p_1	0	0	**1**	1	1	**1**	1
p_2	0	0	1	1	1	1	**2**
p_3	0	0	1	1	**2**	2	2
p_4	0	0	1	**2**	2	2	2
p_5	0	0	1	2	**3**	3	3
p_6	0	0	1	2	3	3	**4**
p_7	0	0	1	**2**	3	3	4

(d)

Dpos

Row	**0**	**1**	**2**	**3**	**4**	**5**	**6**
0	0	7	7	7	7	7	7
1	0	2	7	7	7	7	7
2	0	2	6	7	7	7	7
3	0	2	4	7	7	7	7
4	0	2	3	7	7	7	7
5	0	2	3	4	7	7	7
6	0	2	3	4	6	7	7
7	0	2	3	4	6	7	7

Fig. 2. (a) A toy example of a structure superposition of two proteins p and q. A line connecting p_i and q_j indicates that $\| p_i-q_j \| \le d$ (b) The score matrix $S=S(i,j)$ (c) Dynamic programming matrix, with *k-matches* in bold and *dominant k-matches* underlined (d) Updating the array *Dpos* of positions of dominant matches, row by row.

$M = \bigcup_{k>0} Q_k$, where Q_k is the set of *k-matches*, defined as $Q_k = \{(i,j) \mid (i,j) \in M \text{ and } C[i,j] = k\}$.

To find an optimal alignment $A_d(p,q)$, it is sufficient to focus on *dominant k-matches* [35], i.e. *k*-matches (i,j) such that $j = \min\{j' \mid (i,j') \in M\}$. A single row of the dynamic programming matrix contains at most one dominant *k-match*, for every $k > 0$. An optimal alignment $A_d(p,q)$ corresponds to a sequence of dominant matches, one for each pair of aligned residues (Fig. 2c). To quickly find this sequence, we scan the rows of the score matrix and update the array *Dpos* of positions of dominant matches in q (Fig. 2d). Initially, $Dpos[0]$ is set to zero and all other values are set to n+1. The rows of the score matrix are then processed, one by one, from right to left. Whenever a dominant *k*-match (i,j) is found, $Dpos[k]$ is set to *j*. The array of positions of dominant matches can be efficiently updated using an $O(\log n)$ binary search algorithm. More specifically, for each match (i,j), the binary search algorithm can be applied to determine whether there exists an open interval $(Dpos[k], Dpos[k+1])$ containing j. If such an interval exists, $Dpos[k+1]$ is set to *j*. Since the score matrix contains no more than $K_d \cdot m = O(m)$ matches, all of its rows can be processed in $O(m \log n)$ time. The pseudocode for processing the rows of the score matrix (FORWARD), performing the binary search (SEARCH), and tracing back an optimal alignment (TRACEBACK) are given below.

```
Algorithm FORWARD(L)
// Computes the optimal alignment score bestScore, the array of dominant
// positions Dpos and the array Back for tracing back an optimal alignment
// A_d(p,q).
1.    bestScore←0; Dpos[0]←0
2.    for l←1 to n do
3.        Dpos[l]←n+1
4.    for i←1 to m do
5.        len←length of L[i]
6.        for pos←len downto 1 do
7.            j←L[i,pos]
8.            k←SEARCH(j)
9.            if k != −1
10.               Dpos[k+1]←j
11.               Back[k+1]←i
12.               if k+1>bestScore
13.                   bestScore←k+1
```

```
Algorithm SEARCH(j)
// On input j, returns the integer k, such that Dpos[k]<j<Dpos[k+1], or -1
// if such an integer does not exist.
1.      left ← 0
2.      right ← bestScore + 1
3.      while left ≤ right do
4.           mid ← ⌊(left + right)/2⌋
5.           if j > Dpos[mid]
6.                left ← mid + 1
7.           else if j < Dpos[mid]
8.                right ← mid − 1
9.           else return −1
10.     if j < Dpos[mid]
11.          return mid − 1
12.     else return mid
```

```
Algorithm TRACEBACK
// Computes an optimal alignment A = A_d(p,q).
1.      for k ← bestScore downto 1 do
2.           A[Back[k]] ← Dpos[k]
```

It is not difficult to see that the hidden constant factor in the running time of the above alignment routine is $\sim K_d$, where K_d is the number of C_α atoms that can be packed inside a sphere of radius d in R^3. In practical applications, the hidden constant is small since the distance cutoff is usually set below 8Å.

2.3 Benchmark

To test the efficiency of our algorithm in real applications, we compiled a test set consisting of 246 pairs of structurally related chains (at various structural levels) from the FSSP database [36]. Our test set is chosen so that the protein pairs can be grouped into three bins of equal size (82 pairs in each), according to the chain lengths: $m,n \le 250$, $250 < m,n \le 500$ and $m,n > 500$. The set of pairs of proteins used in our analysis can be downloaded from *http://bioinformatics.cs.uni.edu/fast_align.html*.

Since the efficiency of our method depends on the proteins' geometry and the spatial positions of the proteins relative to each other, we performed a head-to-head comparison of our algorithm and the standard Smith-Waterman algorithm in four different settings. In the first setting (Table 1), we compared the speed of the two methods on a set of pairs of structurally superimposed chains. The chains were optimally superimposed using the MAMMOTH program [37]. The remaining speed

tests, summarized in Tables 2-4, were performed using the same set of protein pairs, but with the chains from each pair positioned randomly in space, instead of being structurally aligned.

We estimated the factor of speedup of our method over the Smith-Waterman algorithm as a function of the distance between the centers of the proteins, c_1 and c_2 using the distance cutoff $d = 3$: $c_1 = c_2$ (Table 2), $\| c_1 - c_2 \| = (r_1 + r_2)/2$ (Table 3) and $\| c_1 - c_2 \| = r_1 + r_2$ (Table 4), where r_1 and r_2 denote the radiuses of the proteins' bounding spheres. We note that, for all practical purposes, the results presented in Tables 2-4 are most relevant, since the majority of superpositions inspected by a typical iterative methods for protein structure matching are far away from an optimal superposition [15,26].

Table 1. Observed factor of speedup of our method over the Smith-Waterman method on the set of structurally superimposed pairs

Chain length:	$m, n \leq 250$	$250 < m, n \leq 500$	$m, n > 500$
Score matrix	2	5	7
Alignment	25	105	176
Total	4	10	13

Table 2. Speedup factor when the structures are randomly oriented but have the same center of mass

Chain length:	$m, n \leq 250$	$250 < m, n \leq 500$	$m, n > 500$
Score matrix	5	12	16
Alignment	88	756	1654
Total	8	22	30

Table 3. Speedup factor on the set of randomly oriented pairs of structures satisfying $\| c_1 - c_2 \| = (r_1 + r_2)/2$

Chain length:	$m, n \leq 250$	$250 < m, n \leq 500$	$m, n > 500$
Score matrix	6	14	18
Alignment	156	860	1709
Total	10	24	34

Table 4. Speedup factor on the set of randomly oriented pairs of structures such that $\| c_1 - c_2 \| = r_1 + r_2$

Chain length:	$m, n \leq 250$	$250 < m, n \leq 500$	$m, n > 500$
Score matrix	8	17	24
Alignment	260	1070	2134
Total	13	33	46

As seen in Table 1, when applied to optimally superimposed chains of lengths $250 < m,n \leq 500$, our alignment method is about 105 times faster than the corresponding Smith-Waterman dynamic programming algorithm. If the cost of computing the score matrix is taken into account, our method is about an order of magnitude faster that the Smith-Waterman algorithm.

We observed a significant increase in the efficiency of our method on structurally unaligned chains, in particular when the structures are far away from each other. For instance, on the set of pairs of proteins of moderate lengths ($250 < m,n \leq 500$), with the same center of mass, the speedup factor is 22 (12 for the score matrix computation and 756 for the alignment). On the other hand, if the bounding spheres of the two structures are only touching each other ($\| c_1 - c_2 \| = r_1 + r_2$), the speedup factor is 33 (17 for the score matrix computation and 1070 for the alignment).

3 Conclusion

Many pairwise structure comparison algorithms minimize the proteins' inter-atomic distances by inspecting many different superpositions of the input structures, keeping track of the best superposition and the alignment found so far. In order to find a solution reasonably close to optimum, these methods must search the space of all superpositions with a fine-tooth comb, performing an alignment procedure each time a new superposition is generated. For this task, even an $O(n^2)$ Smith-Waterman alignment algorithm is computationally too expensive.

We present a much faster algorithm for computing an alignment that maximizes one of the most widely used measures of protein structure similarity, defined as the number of pairs of atoms in two structures that can be fit under a specified distance cutoff. Our algorithm can be readily applied to improve the speed-accuracy tradeoff of many popular protein structure similarity methods, including the methods commonly used in protein structure prediction benchmarks.

References

1. Moult, J., Fidelis, K., Kryshtafovych, A., Rost, B., Hubbard, T., Tramontano, A.: Critical assessment of methods of protein structure prediction Round VII. Proteins 69(S8), 3–9 (2007)
2. Debe, D.A., Danzer, J.F., Goddard, W.A., Poleksic, A.: STRUCTFAST: protein sequence remote homology detection and alignment using novel dynamic programming and profile-profile scoring. Proteins 64, 960–967 (2006)
3. Kim, D.E., Chivian, D., Baker, D.: Protein structure prediction and analysis using the Robetta server. Nucleic Acids Res. 32(suppl. 2), W526–W5331 (2004)
4. Teodorescu, O., Galor, T., Pillardy, J., Elber, R.: Enriching the sequence substitution matrix by structural information. Proteins 54, 41–48 (2004)
5. Zhou, H., Zhou, Y.: Fold recognition by combining sequence profiles derived from evolution and from depth-dependent structural alignment of fragments. Proteins 58, 321–328 (2005)

6. Xie, L., Bourne, P.E.: Detecting evolutionary relationships across existing fold space, using sequence order-independent profile-profile alignments. Proc. Natl. Acad. Sci. USA. 105, 5441–5446 (2008)
7. Gold, N.D., Jackson, R.M.: SitesBase: a database for structure-based protein–ligand binding site comparisons. Nucleic Acids Res. 34, D231-D234 (2006)
8. Poleksic, A., Fienup, M., Danzer, J.F., Debe, D.A.: A different look at the quality of modeled three-dimensional protein structures. J. Bioinform. Comput. Biol. 6, 335–345 (2008)
9. Murzin, A.G., Brenner, S.E., Hubbard, T., Chothia, C.: SCOP: a structural classification of proteins database for the investigation of sequences and structures. J. Mol. Biol. 247, 536–540 (1995)
10. Orengo, C.A., Michie, A.D., Jones, D.T., Swindells, M.B., Thornton, J.M.: CATH-a hierarchic classification of protein domain structures. Structure 5, 1093–1108 (1997)
11. Wu, C.H., Huang, H., Yeh, L.S., Barker, W.C.: Protein family classification and functional annotation. Comput. Biol. Chem. 27, 37–47 (2003)
12. Goldman, D., Papadimitriou, C.H., Istrail, S.: Algorithmic Aspects of Protein Structure Similarity. In: Proceedings of the 40th Annual Symposium on Foundations of Computer Science, pp. 512–522. IEEE Computer Science, Washington, DC (1999)
13. Caprara, A., Carr, R., Istrail, S., Lancia, G., Walenz, B.: 1001 optimal PDB structure alignments: integer programming methods for finding the maximum contact map overlap. J. Comput. Biol. 11, 27–52 (2004)
14. Xu, J., Jiao, F., Berger, B.: A Parameterized Algorithm for Protein Structure Alignment. In: RECOMB, pp. 488–499 (2006)
15. Kolodny, R., Linial, N.: Approximate protein structural alignment in polynomial time. Proc. Natl. Acad. Sci. USA. 101, 12201–12206 (2003)
16. Gerstein, M., Levitt, M.: Using iterative dynamic programming to obtain accurate pairwise and multiple alignments of protein structures. In: Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology, pp. 59–67. AAAI Press, Menlo Park (1996)
17. Levitt, M., Gerstein, M.: A unified statistical framework for sequence comparison and structure comparison. Proc. Natl. Acad. Sci. 95, 5913–5920 (1998)
18. Zhang, Y., Skolnick, J.: TM-align: a protein structure alignment algorithm based on the TM-score. Nucleic Acids Res. 33, 2302–2309 (2005)
19. Pandit, S.B., Skolnick, J.: Fr-TM-align: A new protein structural alignment method based on fragment alignments and the TM-score. BMC Bioinformatics 9, 531 (2008)
20. Oldfield, T.J.: CAALIGN: a program for pairwise and multiple protein structure alignment. Acta Crystallogr. D Biol. Crystallogr. 63, 514–525 (2007)
21. Singh, A.P., Brutlag, D.L.: Hierarchical protein structure superposition using both secondary structure and atomic representations. In: Proceedings of the International Conference of Intelligent Systems in Molecular Biology, vol. 5, pp. 284–293 (1997)
22. Zemla, A.: LGA - a Method for Finding 3D Similarities in Protein Structures. Nucleic Acids Res. 31, 3370–3374 (2003)
23. Siew, N., Elofsson, A., Rychlewski, L., Fischer, D.: MaxSub: an automated measure for the assessment of protein structure prediction quality. Bioinformatics 16, 776–785 (2000)
24. Sali, A., Blundell, T.L.: Comparative protein modeling by satisfaction of spatial restraints. J. Mol. Biol. 234, 779–815 (1993)
25. Ginalski, K., Grishin, N.V., Godzik, A., Rychlewski, L.: Practical lessons from protein structure prediction. Nucleic Acids Res. 33, 1874–1891 (2005)

26. Poleksic, A.: Algorithms for optimal protein structure alignment. Bioinformatics 25, 2751–2756 (2009)
27. Fischer, D., Rychlewski, L., Dunbrack Jr., R.L., Ortiz, A.R., Elofsson, A.: CAFASP3: the third critical assessment of fully automated structure prediction methods. Proteins 53(S6), 503–516 (2003)
28. Rychlewski, L., Fischer, D.: LiveBench-8: the large-scale, continuous assessment of automated protein structure prediction. Protein Sci. 14, 240–245 (2005)
29. Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. 147, 195–197 (1981)
30. Arlazarov, V.L., Dinic, E.A., Kronrod, M.A., Faradzev, I.A.: On economic construction of the transitive closure of a directed graph. Soviet Math. Dokl. 11, 1209–1210 (1970)
31. Masek, W.J., Paterson, M.S.: A faster algorithm for computing string-edit distances. J. Computer and System Science 20, 18–31 (1980)
32. Chamizo, F., Iwaniec, H.: On the sphere problem. Revista Matemática Iberoamericana 11, 417–429 (1995)
33. Hunt, J.W., Szymanski, T.G.: A fast algorithm for computing longest common subsequences. Communications of the ACM 20, 350–353 (1997)
34. Mukhopadhyay, A.: A fast algorithm for the longest-common-subsequence problem. Information Sciences 20, 69–82 (1980)
35. Hirshberg, D.S.: Algorithms for the longest common subsequence problem. JACM 24, 664–675 (1977)
36. Holm, L., Ouzounis, C., Sander, C., Tuparev, G., Vriend, G.: A database of protein structure families with common folding motifs. Protein Sci. 1, 1691–1698 (1992)
37. Ortiz, A.R., Strauss, C.E., Olmea, O.: MAMMOTH (matching molecular models obtained from theory): an automated method for model comparison. Protein Sci. 11, 2606–2621 (2002)