Approximate 3D Motif Search in Proteins with Domain Specific Knowledge

Arno Formella[1], Thorsten Pöschel[2], and Cástor Sánchez Chao[1]

[1] Universidad de Vigo, Spain
Departamento de Informática
formella@uvigo.es
[2] Universität Erlangen–Nürnberg, Germany
Excellence Cluster: Engineering of Advanced Materials
thorsten.poeschel@eam.uni-erlangen.de

Abstract. We present three heuristics including the usage of domain specific knowledge to improve a general purpose algorithm for the 3D approximate point set match problem and its application to the task of finding 3D motifs (like surface patterns or binding sites) in proteins. The domain specific knowledge and further heuristics are used, under certain conditions, to reduce the run time for the search and to adapt the number of reported matches to the expectations of the user. Compared to the general purpose algorithm, the new version is twice as fast, and can be further improved especially for small tolerances in the matches by means of analyzing the distance distributions of the atoms.

1 Introduction

In recent years proteins are compared and analyzed in all of their representations: as primary structures, i.e., their bare sequences of amino acids, as secondary structures, i.e., their principal folds mostly in α–helices and β–sheets, and as ternary structures, i.e., their complete three–dimensional appearances. The definition of similarity in the molecular context always depends on the scientific question under investigation, especially when one takes into account that proteins are not really rigid structures, rather they exhibit certain amount of flexibility due to temperature, solvents, and other factors. In many research areas, such as docking and functional analysis, it becomes more and more important to compare the 3D structure of the molecules [14].

There is quite a range of algorithms and programs available to the scientific community, both commercial and free, that cover different aspects of protein comparison, analysis, and fold prediction, see [6, 10, 12] for comprehensive overviews of different approaches. They include backbone alignment (based on the C_α–chains), secondary structure elements alignment, and sequence–based alignments, which either perform pairwise alignment or multiple structure alignment. Many of them are hosted at publicly accessible web–servers such as [1, 7, 11] and many more.

In this article, we describe an extension of the program **psm** [4]—a program being already included in a public web–based search facility [1] and especially useful to identify small components in large arrangements, for instance, similar surface or binding site structures, loops and hinges etc. within large proteins—with some new features that, under certain conditions, improve its performance and allow for a more adaptive search. The core algorithm of **psm** searches for all occurrences of a small 3D point set, e.g., atoms, in a large 3D point set, e.g., the macro–molecule. Other algorithms dedicated to a similar task include PINTS [13] and needle–haystack [8].

The rest of the article is organized as follows. Section 2 briefly describes the general purpose approximate point set search algorithm. Section 3 introduces the proposed heuristics to improve the run times of the algorithm. Section 4 gathers some preliminary results that we obtained with comparable implementations of all proposed methods. Section 5 states some future actions that can be carried out and, finally, Section 6 summarizes the main contributions of the work.

2 Approximate point set match

Let us, for a moment, forget the specific task that we are searching for 3D motifs in proteins. Rather, we concentrate on the more abstract problem of searching for a small 3D point set within a large 3D point set. Hence, given a small (both in diameter and number of points) set of points as search pattern and a large set of points as search space, find all locations within the search space where the search pattern can be placed best according to a given distance metric and applying a given kind of geometric transformation.

More formally: Let $P = \{p_1, \ldots, p_k\} \in \mathbb{R}^3$, $|P| = k > 1$ be a finite set of 3D points, the search pattern. Let $S = \{s_1, \ldots, s_n\} \in \mathbb{R}^3$, $|S| = n > k$ be a finite set of 3D points, the search space. Find an injective matching function $\mu : P \longrightarrow S$ and a transformation $\tau : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ such that $D(\tau(P), \mu(P))$ is minimum, where D is an appropriate distance measure between the point sets.

Usually, τ is taken as a rigid body transformation, i.e., a combination of a translation and a rotation. Further possibilities include reflection and scaling. Transformations with deformations (e.g., allowing for torsion or hinges) might be considered, too. Common distance metrics include, for instance, root mean square, maximum or average distance:

$$D(\tau(P), \mu(P)) = \sqrt{\frac{1}{k} \sum_{i=1}^{k} |\tau(p_i) - \mu(p_i)|^2}$$

$$D(\tau(P), \mu(P)) = \frac{1}{k} \sum_{i=1}^{k} |\tau(p_i) - \mu(p_i)|$$

$$D(\tau(P), \mu(P)) = \max_{i=1}^{k} |\tau(p_i) - \mu(p_i)|$$

Our algorithm for approximate point set match implemented in **psm** works as follows ([1, 4]): First, we build an undirected distance graph G_P over P, where the nodes of the graph are the points of P and the edges define a connected graph (for simplicity you might assume that G_P is the complete graph). Second, we build an undirected distance graph G_S over S considering only those edges, i.e., distances in S, being similar (see Section 3 below) to the edges in G_P. If the diameter of P is small compared to the diameter of S a regular grid over the search space is used to reduce the number of candidate edges in G_S.

Once the graphs are built, we apply a backtracking algorithm to find all subgraphs of G_S that match G_P both in graph structure and in edge distances. Let us explain the algorithm in some detail. Assume that the nodes in G_P are ordered in such a way that for all p_j, with $1 < j \leq k$, there exists an edge $\{p_i, p_j\}$ with $i < j$. Let G_P^i denote the subgraph induced by the nodes $\{p_1, \ldots, p_i\}$ of G_P; observe that $G_P^k = G_P$. All these k graphs are connected graphs and for each connected graph G_P such a sequence of graphs can be found (if G_P is the complete graph, any sequence of nodes defines such a sequence of connected graphs).

The backtracking algorithms starts with G_P^1 and tries to find iteratively matches for G_P^i in G_S with $1 \leq i \leq k$. Clearly, all nodes in G_S are candidate nodes to be matched to p_1. Now, assume we have already matched a subgraph G_P^i with $i < k$ and let $s_j = \mu(p_j)$, for $1 \leq j \leq i$, denote the nodes of G_S where p_j has been matched. We try to match the next node p_{i+1} in the sequence. All adjacent nodes of s_i that have not been matched so far and that exhibit similar distances to all corresponding nodes with smaller indices are candidates to be matched to p_{i+1}. If such a node of G_S cannot be found, i.e., G_P^{i+1} cannot be matched, G_P^i cannot be extended and hence backtracking takes place: we proceed with the next candidate for p_i. The backtracking algorithm eventually finds all possible matches of G_P in G_S.

For each match we compute the optimal transformation with the help of a direct method for RMS distance metric based on [9] or, for the other distance metrics, with the derivative free optimization algorithm taken from [5].

Let us analyze briefly the run time behavior of the backtracking algorithm. The subgraph matching problem itself is NP–complete. Assuming that the average degree of G_S is d, the worst case run time is $O(n(d \log d)^{k-1}(k-1)!)$, because in each extending iteration at most d neighbors have to be visited and $i-1$ distance checks must be performed. The factor $\log d$ is due to the fact that we have to search for the corresponding edges in G_S. Storing the graph with an adjacency matrix (that would allow for constant time access) rather than using an adjacency list would make the memory requirements for practical cases prohibitive large. Note that G_P, for being assumed to be small, can be stored as adjacency matrix with constant access time for edge queries.

In practical cases, especially when applying the basic point set match algorithm to proteins, a much lower run time can be observed than stated in the worst case analysis. As described in the next section different heuristics can be used

- to reduce the remaining edges in G_S, hence the average degree d is reduced,
- to reorder the nodes in G_P, hence more confining edges are tested early, and
- to exploit problem specific restrictions, hence the number of candidate nodes and/or edges is reduced.

3 Search heuristics

Before going into details, let us first take a look at the similarity relation between two distances. We say a distance e is similar to a distance d whenever $e \in [d(1-\epsilon), d(1+\epsilon/(1-\epsilon)]$ with $0 \leq \epsilon < 1$. The upper bound can often be decreased to $d(1+\epsilon)$ in the case the symmetry of the search is not an issue.

The tolerance ϵ, with $0 \leq \epsilon < 1$, is a user provided value that determines to what extend the point set found in S can deviate from the pattern P. With $\epsilon = 0$ perfect matches are searched for. In [1, 4] ϵ was a fixed value for all edges. In the new implementation, we allow for a more flexible approach taking into account properties of a protein.

We present three heuristics to reduce the search time:

- One that works with the complete graph over the search pattern and reduces the number of remaining edges in G_S taking into account the distribution of distances being present both in P and S.
- One that uses different values for ϵ in the similarity relation depending whether the atoms defining the edge are located close or far on the protein chain.
- One that takes into account that certain chemical bonds within the protein are unlikely to be very flexible.

3.1 Rare distances first

Let the graph over the search pattern be complete. Once all distance intervals over the graph of the search pattern G_P are computed, we compute the distribution of those edges in G_S that fall into the intervals. The interval where the least number of edges of G_S are counted defines the, let's say, rarest distance. Clearly, we need to maintain in G_S only those edges that correspond to the rarest distance and all adjacent edges to their end nodes with the further restriction that the other end must have an edge with appropriate distance to the other end of the rare edge.

We compute the final graph G_S in four steps. First, we determine the distribution to find the rarest distance. Second, we mark all nodes in G_S that are adjacent to an edge similar to the rarest distance. Third, we mark all nodes in G_S that are simultaneously adjacent to the two end nodes of a rare edge. Note that the marks in the steps two and three are different. Fourth, we generate as graph G_S the graph induced by the edges between marked nodes.

With this possibly smaller graph, the substructure search with the backtracking algorithm is performed.

3.2 Flexible distance intervals

Proteins consist of one or various folded chains built as a sequence of amino acids. Usually, the farer two atoms are located on the chain, the larger the tolerance for their distance can be, because proteins are, to a certain degree, flexible structures. To take advantage of this fact in our algorithm, we use this knowledge to define different tolerances for different edges in the graphs.

We present a still very simple method to define such a flexible tolerance. Let Δ be the distance in amino acid count of two atoms in the search space, i.e., their sequence distance, often notated as $|i - j|$ for atoms i and j of the data set. Let M be the total number of amino acids in the search space. We define

$$f = \frac{\epsilon_f \cdot \Delta}{M - 1} + 1$$

as factor to enlarge the tolerated distance for a match of an edge, i.e., we use instead of ϵ the value $f\epsilon$ in the similarity relation. The quantity $\epsilon_f \geq 0$ denotes a user specified value where care must be taken of that $f\epsilon$ does not exceed 1. To allow for a more user friendly way to introduce the additional tolerance ϵ_f with its influence on ϵ, we ask the user for a value $k \geq 1$ and compute

$$\epsilon_f = \left(1 - \frac{1}{k}\right) \cdot \left(\frac{1}{\epsilon} - 1\right)$$

or set $\epsilon_f = 0$ for $\epsilon = 0$. Hence, a value of $k = 1$ means no flexible tolerance, and the larger the value of k the larger the flexible tolerance will be. In other words, for $\epsilon_f = 0$ the search algorithm uses for all edges the same tolerance ϵ as stated above; the same is true for $\epsilon_f = 1$ and atoms belonging to the same amino acid; for other values of ϵ_f and/or atoms not belonging to the same amino acid, ϵ is enlarged with the factor f.

3.3 Exploitation of domain specific knowledge

According to [2] we take into account the mean distance for certain chemical bonds within the backbone of a protein. In more detail, the C–N distance in a peptide bond is typically 1.32 Å, the C–O distance in a carbon–oxygen bond is typically 1.24 Å, the C_α–C distance in a carboxyl group is typically 1.52 Å, and the C_α–N distance is typically 1.45 Å. Analyzing several proteins from the PDB suite (RCSB Protein Data Bank) [3], we confirmed the given mean values, and observed further a 2.232%, 2.3%, 2.332%, and 3.255% maximum increase/decrease in distance for these four chemical bonds.

We use a simple table look–up method to identify the corresponding value for the tolerance ϵ depending on the types of the atoms and their distance in amino acid count. The C_α–N bond is only present when both atoms belong to the same amino acid, the same is true for both the C–O and the C_α–C bonds. The C–N bond is present when the amino acid distance of the two participating atoms is one. To build the look–up table we assign to each atom an index i_a according to Table 1.

atom	any	C_α	N	C	O
i_a	0	2	4	8	14

Table 1. Atom indices according to their type.

If the amino acid distance Δ between the two atoms A_0 and A_1 defining an edge in the distance graph is 0 or 1, we compute a table index $i = \Delta + i_a(A_0) + i_a(A_1)$ which is used to address a table with 30 entries (see Table 2).

i	0–5	6	7–9	10	11–12	13	14–21	22	23–29
ϵ	0	0.03255	0	0.0232	0	0.2232	0	0.023	0

Table 2. Tabularized values for ϵ for pairs of atoms.

If the encountered entry in the table is 0, then we use the ϵ value given by the user (possibly taking into account the factor f as described in the previous section), if it is not 0, the table entry without modification is used as ϵ value for the specific edge.

4 Results

We present some preliminary results illustrating the improvements comparing the new version with the different improvements to the original **psm** software. The run times have been obtained in [seconds] on a GNU/Linux system with a Q9550 Intel processor at 2.83 GHz. **psm** is still implemented as a single threaded application.

As search space we used the protein 1IRU (hydrolase, mammalian proteasome) [15] with 47 589 atoms and an active site of the proteasome (see Figure 1). The active site contains 40 atoms distributed over at least four non–consecutive amino acids. The active site is a cutout of the proteasome which appears with certain modifications several times in the protein (see Figure 2). Whenever two matches in the search space share an atom, we consider the two matches belonging to the same cluster. **psm** reports to the user both the number of matches and the number of clusters encountered. A cluster shows where the motif is located in the protein, whereas the match itself shows in what constellation (and quality according to some pairwise distance function) the motif is present.

Figure 3 shows the preprocessing times (left plot) and the search times (right plot) for the **psm** version without any improvement (upper curves) and the same for the version where the domain specific knowledge has been taken into account, i.e., the distances between certain pairs of atoms as stated in Section 3.3 vary at most up to the tabularized percentages. The plots are drawn over the distance tolerance ϵ from 0 to 35%, i.e., the mismatch of the inter–atom distances

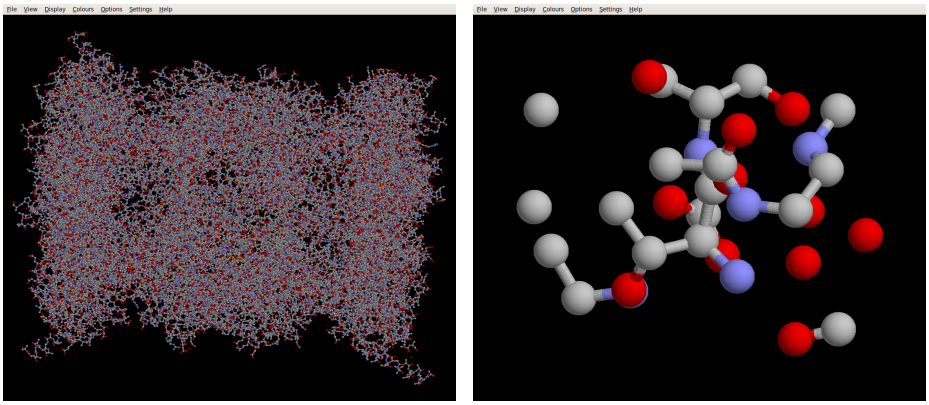

Fig. 1. Search space (proteasome) and search pattern for the test cases.

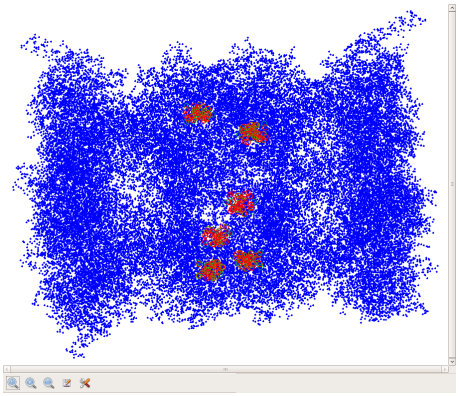

Fig. 2. Clusters found with **psm**.

comparing the search pattern to the encountered location in the search can be that large. As one can observe the preprocessing time is almost the same for both cases, however, the search time of the new version is half of the time of the old version.

If we look at the number of matches found by the two versions (see Figure 4), we see that including domain specific knowledge reduces the number of reported matches (note the logarithmic scale in the plot). However, it is important to mention that, at least in our current tests, the number of clusters and their location is for all values of ϵ exactly the same. We have observed the same behavior for all other tests described in the on–going.

Figure 5 shows the preprocessing times (left plot) and the search times (right plot) for versions of **psm** taking into account the distance distributions, i.e., searching for the rarest distance first. The upper curve does not use domain specific knowledge. The plots are drawn over the distance tolerance ϵ from 0 to

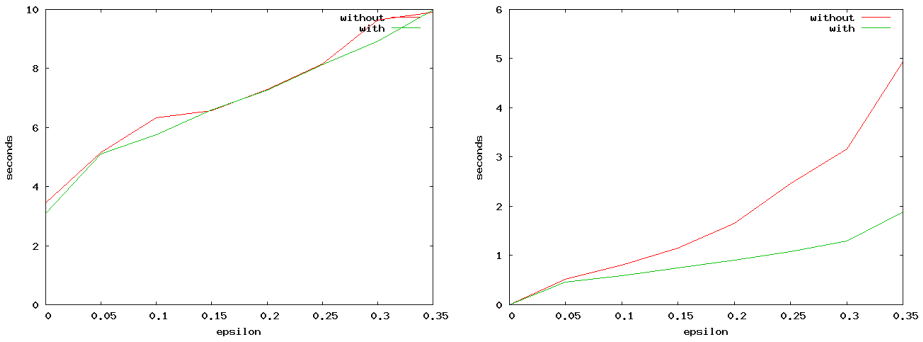

Fig. 3. Preprocessing times (left plot) and the search times (right plot) for the **psm** version without any improvement (upper curves) and the same for the version where the domain specific knowledge has been taken into account.

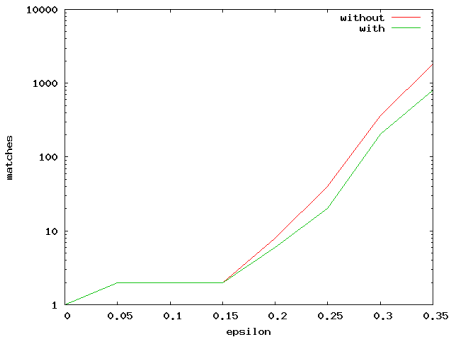

Fig. 4. Number of matches for the two versions without (upper curve) and with (lower curve) domain specific knowledge (note the logarithmic scale).

20%, i.e., the mismatch of the inter–atom distances comparing the search pattern to the encountered location in the search can be that large. As one can observe the preprocessing time is almost the same for both cases, but has increased about 20% compared to the versions without consideration of the distribution. The search time with use of domain specific knowledge again is, at least for larger values of ϵ, almost halved. Moreover, if one compares the search times shown in Figures 3 and 5, searching for the rarest distance first, reduces the search time slightly, especially for small tolerances. Hence, although the preprocessing time has been increased due to the processing of the distribution and the more complex graph construction, the faster search time may lead even to an overall faster algorithm, when partial matches (leaving several atoms out) have to be encountered.

Figure 6 shows the effect of introducing flexible distances according to the amino acid distance of the atoms (as introduced in Section 3.2). The preprocessing times (left plot) are almost the same as in the case of no flexible distance

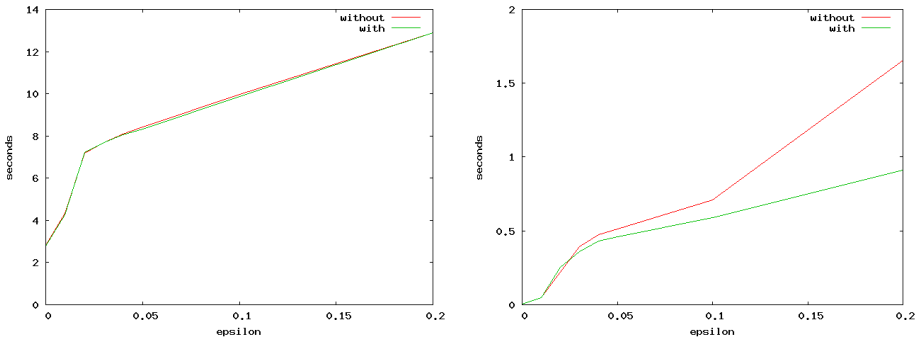

Fig. 5. Preprocessing times (left plot) and the search times (right plot) for versions of **psm** taking into account the distance distributions. The upper curve does not use domain specific knowledge, whereas the lower uses that knowledge.

(compare to left plot in Figure 3). The run times have increased considerably as expected, because we set $k = 1000$, a rather large value, i.e., the distance tolerances for widely separated atoms according to their amino acid distance became quite large. The important fact to notice is that the introduction of domain specific knowledge (lower curves) allows for much faster run times for larger tolerance ϵ.

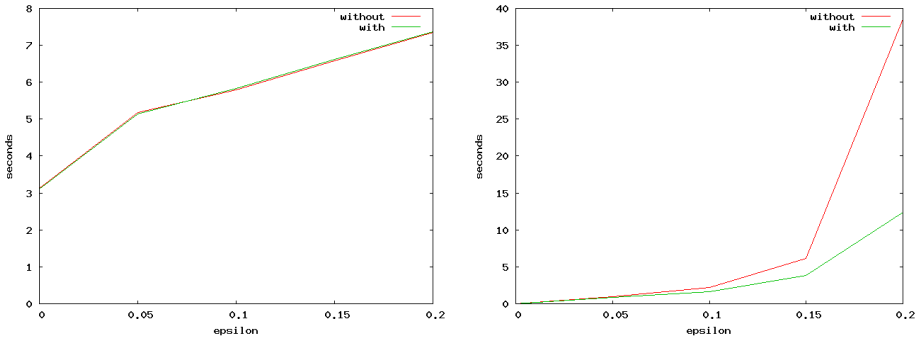

Fig. 6. Preprocessing times (left plot) and the search times (right plot) for versions of **psm** taking into account flexible distances. The upper curve does not use domain specific knowledge, whereas the lower uses that knowledge.

Figure 7 shows what happens if both flexible distances and analysis of the distance distribution is taken into account. Again preprocessing times (left plot) with and without domain specific knowledge is almost equal and more or less the same when not dealing with flexible distances (compare to left plot in Figure 5). The search times (right plot) show the same behavior as seen in Figure 6,

i.e., with flexible distances it is not worthwhile to search with the rarest distance first, at least, for large values of the tolerance ϵ.

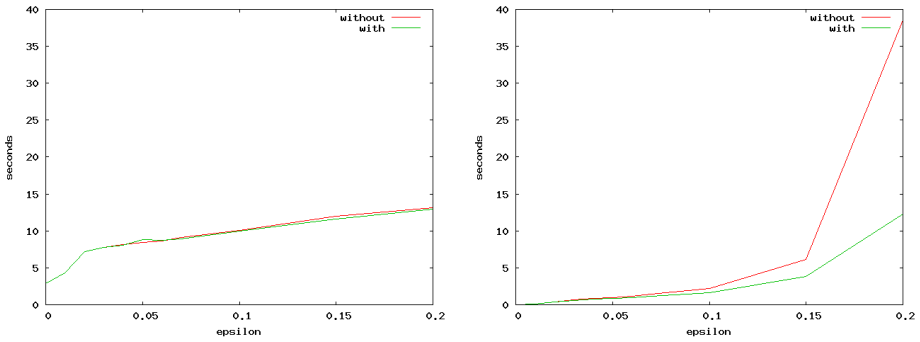

Fig. 7. Preprocessing times (left plot) and the search times (right plot) for versions of **psm** taking into account both flexible distances and analysis of the distance distribution. The upper curve does not use domain specific knowledge, whereas the lower uses that knowledge.

Preprocessing takes somewhat longer but searching is much faster. So it pays off, for instance, when searching for more than one pattern at a time, or when searching for partial matches leaving out one or several atoms.

5 Further work

The ideas and preliminary tests as presented in this article can be extended taking into account the following:

- Up to now we use only domain specific knowledge of the backbone of the protein. Possibly it would be useful to take into account further structures, such as rings, in residues which exhibit less flexible distances as well.
- We might improve the simple amino acid distance dependent tolerance to a more sophisticated one, especially considering the possible rotation of a triple of amino acids. Moreover, the dependence of the overall amino acid count of the protein should be eliminated, i.e., the value of M should be established to a fixed value gathering statistics calculated from PDB structures.
- During the backtracking we might consider the rotational restrictions for certain atom arrangements to early discard candidates.
- The user might introduce individual distance tolerances for certain or all distances within the search pattern.
- Clearly, more rigorous statistical evaluation of the proposed methods on a sufficiently large and diverse set of search spaces and motifs must be elaborated.

- Other type of domain specific knowledge might be introduced, for instance, removal of interior atoms whenever a surface pattern is searched for, matching of only chemically similar atoms, or incorporating the residue type restrictions taking into account properties like electrical charge or hydrophobicity.
- The search algorithm should be parallelized to run at least with a small number of threads taking advantage of modern processor architectures.

6 Conclusion

We presented three approaches and certain details of their implementation together with some preliminary results on how to improve the performance of a general purpose approximate point set match algorithm in the field of structural protein analysis. Introducing domain specific knowledge, such as less flexible chemical bonds, reduces the run time of a search to the half without increase in the preprocessing time. The modified algorithm finds the same locations in terms of clusters as the general algorithm. The run time of a search can be further reduced slightly taking into account the distance distribution in the search graphs. However, the preprocessing time is increased by roughly 20%. Hence, analyzing the distributions pays off whenever the preprocessing time can be amortized, e.g., if partial matches must be found. With the help of flexible distances according to the amino acid distance of the atoms, large tolerances can be employed with moderate increase of the overall search time.

References

1. R.A. Bauer, P.E. Bourne, A. Formella, C. Frömmel, C. Gille, A. Goede, A. Guerler, A. Hoppe, E.W. Knapp, T. Pöschel, B. Wittig, V. Ziegler, and Preissner R. Superimposé: a 3d structural superposition server. *Nucleic Acids Research*, 36 (Web Server Issue):W55–W59, 2008.
2. J.M. Berg, J.L. Tymoczko, and L. Stryer. *Biochemistry.* W.H. Freeman&Co Ltd., 2006.
3. H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.
4. A. Formella. Approximate point set match for partial protein structure alignment. In F.M. Couto, J.S. Silva, and P. Fernandes, editors, *Proceedings of Bioinformatics: Knowledge Discovery in Biology (BKDB2005)*, pages 53–57. Facultade Ciencias Lisboa da Universidade de Lisboa, ISBN 972-9348-12-10, June 2005.
5. U. García-Palomares and J. Rodríguez. New sequential and parallel derivative–free algorithms for unconstraint optimization. *SIAM Journal of Optimization*, 13:79–96, 2002.
6. Anthony D. Hill and Peter J. Reilly. Comparing programs for rigid-body multiple structural superposition of proteins. *Proteins: Structure, Function, and Bioinformatics*, 64(1):219–226, 2006.
7. L. Holm, S. Kääriäninen, P. Rosenström, and A. Schenkel. Searching protein structure databases with dalilite v.3. *Bioinformatics*, 24:2780–2781, 2008.

8. A. Hoppe and C. Froemmel. Needle-haystack: A program for the rapid recognition of local structures in large sets of atomic coordinates. *Journal of Applied Crystallography*, 36:1090–1097, 2003.
9. W. Kabsch. A discussion of the solution for the best rotation to relate two sets of vectors. *Acta Crystallographica A*, 34:827–828, 1978.
10. R. Kolodny, P. Koehl, and M. Levitt. Comprehensive evaluation of protein structure alignment methods: scoring by geometric measures. *Journal of Molecular Biology*, 346:1173–1188, 2005.
11. R. Maiti, G.H. Van Domselaar, H. Zhang, and D.S. Wishart. Superpose: a simple server for sophisticated structural superposition. *Nucleic Acids Research*, 32 (Web Server Issue):W590–W594, July 2004.
12. M. Novotny, D. Madsen, and G.J. Kleywegt. Evaluation of protein fold comparison servers. *Proteins*, 54:260–270, 2004.
13. A. Stark, S. Sunyaev, and R.B. Russell. A model for statistical significance of local similarities in structure. *Journal of Molecular Biology*, 326:1307–1316, 2003.
14. M. Thimm, A. Goede, S. Hougardy, and R. Preissner. Comparison of 2d similarity and 3d superposition. application to searching a conformational drug database. *Journal of Chemical Information Computing Science*, 44:1816–1822, 2004.
15. M. Unno, T. Mizushima, Y. Morimoto, Y. Tomisugi, K. Tanaka, N. Yasuoka, and T. Tsukihara. The structure of the mammalian 20S proteasome at 2.75 A resolution. *Structure*, 10(5):609–618, 2002.