FlexProt: an Algorithm for Alignment of Flexible Protein Structures

Maxim Shatsky[1] **Ruth Nussinov**[2,3*], **Haim J. Wolfson**[1 †]

[1]Dept. of Computer Science, School of Math. Sc., Tel Aviv University, Tel Aviv 69978, Israel,
Telefax : +972-3-640 6476, e-mail : wolfson@math.tau.ac.il;
[2]Sackler Inst. of Molecular Medicine , Sackler Faculty of Medicine , Tel Aviv University.
[3] IRSP - SAIC, Lab. of Experimental and Computational Biology, NCI - FCRDC,
Bldg 469, Rm 151, Frederick, MD 21702, USA

Abstract

We present a novel technique for the alignment of flexible molecules. The method does not require *a priori* knowledge of the flexible regions. It is based on 3-D pattern matching methods combined with a graph theoretic technique and it is not sensitive to insertions and deletions. The FlexProt algorithm simultaneously detects the hinge regions and aligns the rigid subparts of the molecules. It does it by efficiently detecting maximal congruent rigid fragments in both molecules and calculating their optimal arrangement which does not violate the protein sequence order. The algorithm is highly efficient, it performs structural comparison of a pair of two hundred amino acid proteins in about 7 seconds on a standard desktop PC. Extensive experiments were performed and preliminary results are presented.
Keywords: flexible protein structure alignment, hinge detection, graph-theoretic algorithms, minimal RMSD alignment, detection of conformational changes.

Introduction

Alignment of protein sequences has become one of the major indispensable tools in bioinformatics (Gusfield 1997). Nevertheless, it is well know (Branden & Tooze 1991) that protein structure stores significantly more information than its sequence. In particular, structure has been much better conserved than sequence during evolution. Thus, one would expect that protein structural alignment algorithms could supply significant information which cannot be received from sequence alignment alone. While the number of known protein structures was small, such structural alignments were mainly detected by visual inspection. In the last decade, with the significant increase in available protein structures, the need for highly efficient structural alignment methods has become evident. This need will become more acute with the advance of the Structural Genomics effort. Structure alignment algorithms also have critical applications for Computer Assisted Drug Design, where one is interested to align ligands acting on a similar receptor. A recent comprehensive survey of these methods appeared in (Lemmen & Lengauer 2000).

Numerous methods have been proposed to tackle the protein structural alignment task, when both proteins are viewed as rigid objects. A small sample includes (Taylor & Orengo 1989; Vriend & Sander 1991; Mitchel *et al.* 1989; Nussinov & Wolfson 1991). Many of them exploit the amino acid sequence order, which enables them to apply techniques which are close in spirit to sequence alignment methods, such as the dynamic programming (e.g. (Taylor & Orengo 1989).) Sequence order independent structural comparison methods, such as *Geometric Hashing* (Nussinov & Wolfson 1991), have a wider range of applications and can also be used to the structural comparison of protein-protein interfaces and docking.

However, in general proteins cannot be viewed as rigid objects, but rather as flexible objects composed of rigid parts with intermediate flexible regions. A well known survey of domain movements in proteins was published by (Gerstein, Lesk, & Chothia 1994). Two predominant motions of neighboring protein domains relative to each other have been detected - *hinge* motion and *shear* motion. The hinge motion resembles rotational joints defined in Robotics, while shear motion can be described as the movement of parallel plates.

There are relatively few methods that deal with the structural alignment of flexible proteins. The method of (Verbitsky, Nussinov, & Wolfson 1999) which is based on an extension of the Generalized Hough Transform method (Wolfson 1991), and was also applied to flexible docking by (Sandak, Nussinov, & Wolfson 1995), aligns a flexible protein with a rigid one assuming prior knowledge of the hinge locations on the flexible protein.

More general method, **FlexMol** algorithm, for structural alignment of flexible molecules was presented in (Shatsky *et al.* 2000). The **FlexMol** algorithm is fully sequence order independent, however, it does require *a-priori* input of the hinge positions.

Methods originally developed for the alignment or

†Corresponding author - H.J. Wolfson, e-mail : wolfson@math.tau.ac.il.

docking of flexible drug molecules (Rarey *et al.* 1996; Rigoutsos, Platt, & Califano 1996) also require *a priori* knowledge of the hinge locations.

Here we present an algorithm, **FlexProt**, for the alignment of a rigid molecule with a flexible one which does not require any *a priori* knowledge on the partition of the flexible molecule into rigid parts. The algorithm automatically detects the hinge regions. The input to this algorithm is the rigid molecule and some conformation of the flexible molecule. The output is a list of best alignments ranked according to the size of the overall alignment, the size of the individual rigid parts, the RMSD of the alignment and some additional criteria which are outlined below. The program simultaneously discovers both the matching rigid fragments and the intermediate flexible regions for each candidate solution. This new method accomplishes a much more complex tasks than the previous flexible alignment methods, where the flexible molecule partition was required in advance. Yet, in order to accomplish this task efficiently it has to exploit the amino acid sequence order and is not sequence order independent as (Verbitsky, Nussinov, & Wolfson 1999) or as **FlexMol** (Shatsky *et al.* 2000).

We have done numerous experiments which are summarized in the "Experimental Results" section. The method proved to be robust and highly efficient comparing a pair of flexible structures of about 200 amino acids each in average time of 7 seconds on a standard desktop PC (400MHz, 256 Mb RAM). **FlexProt** algorithm has been implemented in C++. The results represent our preliminary attempts to tackle the flexible alignment task without *a priori* knowledge of the flexible regions. Various improvements and additional experimentation are planned in the near future.

The Flexible Protein Alignment and Hinge Detection Algorithm (FlexProt)

In this section we outline our algorithm for the alignment of flexible protein structures and automatic detection of the intermediate flexible regions.

Input. The input to the algorithm are two protein molecules M_1 and M_2, each being represented by the sequence of the 3-D coordinates of its C_α atoms, $M_1 = v_1, ..., v_n$ and $M_2 = w_1, ..., w_m$. These sequences are ordered according to the standard amino acid order of the protein backbone. In case a protein is composed of several chains, the algorithm treats each chain separately in the pairwise comparison. In the sequel we shall use the term *molecule* for a single chain.

Task. The goal of our flexible structural matching algorithm is to decompose the two molecules into a minimal number of disjoint fragments of maximal size, such that the matched fragments will be almost congruent [1].

[1] Two fragments are congruent, if they have the same number of C_α atoms and there exists a 3-D rotation and translation which superimposes the corresponding atoms with a small RMSD.

The matching fragment arrangement should be consistent with the order of the molecular chain. The regions between the fragments are flexible regions. Actually, we aim to solve the more complex *partial alignment* task, where we require that only large enough flexible substructures of the molecules be aligned (analogous to the "ends-free" alignment of protein sequences). Obviously, due to the inherent inaccuracies in approximate matching and the tradeoffs between the accuracy of matching and the size of the matching fragments, one cannot expect to find an 'optimal' solution. The task is, therefore, to output a set of candidate solutions sorted according to the above mentioned criteria and user imposed accuracy thresholds.

Assume that molecule M_1 has undergone a number of hinge bendings and other structural changes. Yet, we assume that between the flexible regions there are still a number of fragments without significant change in their 3-D structure. Denote the resulting molecule by M_2. Under our assumptions there exists a set of rigid fragments of M_2 that can be matched (is congruent) with the proper set of fragments of M_1 . The model presented applies not only to different conformations of a given molecule, but also to the general case of flexible motif detection.

The **first step** of our algorithm detects candidate 'big enough' congruent rigid fragments which satisfy the above mentioned conditions. Once such fragment pairs are detected, we wish to find an optimal subset that describes a possible conformation of M_2 with respect to M_1. The subset that we seek is, actually, a sequence of disjoint fragments that follows the sequence of the C_α atoms of M_1 and M_2. This task is accomplished by the **second step** of the algorithm. One possible solution for this task is to enumerate over all subsets of the matched fragment pairs and to choose the best scoring ones which are consistent with the amino acid sequence order. However, this would be extremely inefficient. If N is the number of matched fragment pairs and k the length of the shorter chain, then the complexity might be as large as $O(\binom{N}{k})$, i.e. grow exponentially with the size of the input.

We have adopted a solution which is based on the efficient *"Single-source shortest paths"* algorithm in directed weighted acyclic graphs (DAG). As we show in section "Joining Rigid Fragments" the solution of our task can be reduced to the the detection of shortest paths from a single source in the directed graph $G = (V, E)$, where the vertices V are the congruent fragment pairs. As is well known (see e.g.(Cormen, Leiserson, & Rivest 1990)) the complexity of this task is $O(|E| + |V|)$.

The **third step** of the algorithm clusters the consecutive fragment pairs that have a similar 3-D transformation. Assume that M_1 and M_2 have a structurally similar β-sheet. It is likely that the "turn" regions connecting the β-strands will not be congruent in M_1 and

M_2. Our algorithm will detect the similarity among the β-strands of the different molecules, yet it might view them as disjoint congruent pairs, due to the structural dissimilarity of the "turns". However, except for the "turns" the whole β-sheet may match with the same 3-D transformation. We would like to consider it as a single congruent region pair. Thus, we cluster consecutive congruent pairs, which share an almost similar transformation.

Therefore, the final *partial alignment* solution is represented by a number of congruent region pairs, each of them composed of one or several congruent fragment pairs.

Detection of Congruent Rigid Fragment Pairs

In this step we detect the almost congruent fragments of M_1 and M_2. Namely, we search for pairs of equal size fragments, one from each molecule, so that there exists a 3-D rotation and translation of one of the fragments that superimposes it on the other with small RMSD (the RMSD threshold is a user defined parameter). There are two aspects to the fragment pair detection task :

- (i) **Correspondence** - detection of candidate (almost) congruent fragment pairs. In particular, this means the detection of those atom pairs which can be superimposed with a small RMSD. We nickname this corresponding atom pairs list - a **match-list**.
- (ii) **Superimposition** - given the corresponding C_α atom sets (match-list) find the rotation and translation which superimposes these sets with minimal RMSD.

The superimposition aspect of this task has been dealt with intensively and many efficient solutions can be found both in the Computer Vision and Structural Biology literature. We have used the algorithm of (Schwartz & Sharir 1987) whose complexity is linear in the size of the matched C_α atom pairs.

Let us consider the **correspondence** task.

Assume, first, that we have a single matching atom pair (v_a, w_b), where $v_a \in M_1$ and $w_b \in M_2$. Then, we iteratively try to extend this initial match-list by adding one more atom pair to the left and to the right (following the backbone direction) of the current match-list till we obtain the longest pair of consecutive congruent fragments which includes our initial matching atom pair. The congruence of the fragments is ensured by the fact that the RMSD of the rotation and translation giving the best superimposition can be derived based on the match-list alone (see (Schwartz & Sharir 1987) and Appendix A. Moreover, this RMSD can be continuously updated by a constant number (O(1)) of operations at each step. Thus, one can proceed iteratively till the RMSD of the best match exceeds a previously defined threshold. At the end of this iterative extension process, we obtain a match-list :

$(v_i, w_j), (v_{i+1}, w_{j+1}), ..., (v_a, w_b), ..., (v_{i+l}, w_{j+l})$.

Namely, fragment $(v_i, ..., v_{i+l})$ of M_1 is almost congruent to the fragment $(w_j, ..., w_{j+l})$ of M_2 . If the length of the matching fragments is above a predefined threshold $MinFragSize$, we define the matching fragment pair as $F_i^1 F_j^2(l)$.

Given $F_i^1 F_j^2(l)$ the next alignment is initiated at $(v_{i+(l+1)}, w_{j+(l+1)})$ and the same procedure is repeated. The process can be viewed as moving along the diagonals of the matrix M_{nm} which represents the indices of M_1 and M_2. The indices of the initial starting atom pairs are the starting points of the diagonals of the matrix M_{nm}.

Since the update of the RMSD at each step is $O(1)$, the complexity of finding a particular matching fragment pair $F_i^1 F_j^2(l)$ is linear in the length of the fragments, namely $O(l)$. Since matching fragment pairs may overlap, each matching pair of atoms can participate in several match-lists. In the worst case there can be $O(k)$ such match-lists ($k = max(|M_1|, |M_2|)$), although in practice it is $O(1)$. Thus, the overall complexity of the first step of the algorithm is $O(k^3)$ in a worst case estimate and $O(k^2)$ in practice.

Joining Rigid Fragments

Once we have a set of congruent fragment pairs $F_i^1 F_j^2(l)$ we wish to find an optimal subset of it, which describes a possible alignment of M_2 with M_1 allowing flexibility in M_2 between the fragments F_j^2. Ideally, this alignment should include a sequence of disjoint congruent fragment pairs in ascending order of the indices i and j. Practically, one should allow certain overlap of consecutive fragments on the same chain. Our method is close in spirit to the one implemented in FASTA as described in (Gusfield 1997).

In the nutshell the idea of this phase of our algorithm is as follows :

1. Represent congruent fragment pairs as vertices of a graph.
2. Join two vertices by a directed edge, if the fragment pairs that they represent might be consecutive in the final alignment. The result is an acyclic directed graph.
3. Assign weights (penalties) to the edges, rewarding long matching fragments and penalizing big gaps (insertions/deletions) as well as large discrepancies in the relative number of insertions and deletions in both proteins.
4. Apply the *"Single-source shortest paths"* algorithm to the weighted graph.
5. Collect the paths found by the *"Single-source shortest paths"* algorithm according to the number of vertices in each path, sort these candidate solutions according to the total size and RMSD of the fragment alignment and output the best scoring hypotheses. Each such hypothesis represents a sequence of consecutive congruent fragment pairs.

To properly implement the above mentioned general scheme one has to resolve several issues which are outlined below.

Let us define the weighted graph $G = (V, E)$ where the vertices $V = \{F_i^1 F_j^2(l)\}$ are the congruent fragment pairs. A directed edge between the pair of vertices $F_i^1 F_j^2(l)$ and $F_{i'}^1 F_{j'}^2(l')$ is defined if the following conditions hold :

- The fragments are in ascending order. Namely, $i < i'$ and $j < j'$.
- The gaps (insertions/deletions) between consecutive fragments are limited by user defined thresholds MaxIns and MaxDels.

The previous phase of the algorithm, which computes congruent fragment pairs, may produce overlapping fragments representing different vertices in our graph. Although we do not want to exclude edges between such fragments in the graph, since they might contribute to the "correct" solution, we have to introduce a restriction on the fragment overlap. Also, in later stages of the algorithm, we update consecutive overlapping fragments and make them disjoint by equally dividing the overlap region between them. Thus the following restrictions should be introduced as well :

- The possible overlap between consecutive fragments is less than 60% of the smaller fragment size.
- If there has been an overlap between fragments, then the original fragment length without half of the maximal overlapping interval should still be above the minimal fragment length requirement.

Next, we define the weight of an edge as:

$$w(e) = -((l+1) - \lceil \Delta \rceil)^2 + max(|Ins|, |Dels|)$$

$$+||Ins| - |Dels||,$$

where Δ is half of the maximal overlapping interval (if there is no overlap, then Δ is zero). Notice that the weight is independent of the size of $F_{i'}^1 F_{j'}^2(l')$. In this weight function we reward quadratically the size of $F_i^1 F_j^2(l)$ (negative weight), and introduce penalty for large insertions and deletions. The third factor gives priority to the edges with a smaller (absolute value) difference between Ins and $Dels$.

In this manner we build a weighted directed acyclic graph (DAG). A shortest weighted path in this graph should ideally correspond to a consecutive alignment of congruent matching fragments, which are relatively long and have relatively similar numbers of gaps in the non-aligned regions.

In order to find efficiently the shortest paths we add a virtual vertex, which we call the *starting vertex*, to the graph G and connect it to every $F_i^1 F_j^2(l)$ assigning a zero weight to these connecting edges.

Next we run the *"Single-source shortest paths"* algorithm from the *starting vertex*. At the end of the algorithm for each vertex v of the graph G we know the shortest path from the *starting vertex* to the vertex v. Each such path may represent a possible solution to the problem. The paths differ by the number of vertices (or rigid fragments). Since we do not know *a priori* the exact number of rigid fragments which represent the possible conformation of M_2 with respect to M_1 we collect all the paths (the number of these paths is $|V|$). Then, we divide them according to the number of vertices, i.e. we obtain the set $\{S_i\}$ where S_i contains all the paths which are of i vertices length.

From now each path represents some conformation of molecule M_2 with respect to molecule M_1. Note, that only C_α atoms belonging to the rigid fragments participate in the conformation. We calculate the RMSD of the total matching set, which includes all the matched fragments. Notice, that the total RMSD cannot exceed the maximum RMSD of the fragment pairs. Then we sort each set S_i according to the total size and RMSD of it's matching sets. Thus, picking the best, let's say, 10 results from each set S_i, will give us a number of possible solutions to the problem.

The complexity of the second step of the algorithm depends on the size of the graph G. If $n = |M_1|$ and $m = |M_2|$ then the number of vertices $|V| \leq n * m$, and the number of edges $|E| \leq (n * m)^2$. The time complexity of the *"Single-source shortest paths"* algorithm is linear in the size of the directed acyclic graph.

Thus the running time of the second step is bounded by $O((n * m)^2)$.

Clustering.

Each shortest path found in the previous step represents a sequence of congruent matching fragments which are divided by regions which could not be matched. Each pair of congruent matching fragments defines a 3-D rigid transformation (rotation and translation) which superimpose them with minimal RMSD. Some of these rigid transformations might be practically identical, allowing us to group several congruent fragment pairs into congruent rigid domains[2]. This can happen, for example, in a helix-turn-helix motif where the helices are in the same rigid configuration, yet the turns are different. The aim of the clustering step is to detect consecutive congruent fragment pairs that share approximately the same transformation and to join them into congruent rigid domain pairs.

For each path in the graph G, which is the output of the previous stage, the following fragment clustering procedure is done. The first congruent fragment pair (first vertex of the path) is a singleton cluster. Then, we take the next congruent fragment pair (second vertex of the path) and check whether there is a rigid transformation which superimposes simultaneously both fragment pairs with an RMSD, which is below our threshold. If successful we go to the next fragment pair, check

[2] A domain here is not necessarily the classical biological definition of a domain.

whether it can be joined to the current cluster and so on. If we fail to join a congruent fragment pair, then the current cluster is defined as a congruent rigid domain, and one starts a new cluster with the fragment pair that failed to join the previous cluster. This procedure is continued till we exhaust all the congruent fragment pairs. Now, each path is divided into a number of congruent rigid domain pairs each of them composed of one or several congruent fragment pairs. Each congruent rigid domain pair is characterized by a single 3-D rotation and translation which superimposes the domain of molecule M_1 onto its "twin" domain in molecule M_2 with a small RMSD.

The time that takes to decompose a path into rigid domain pairs is proportional to the total size of the fragments included in the path, which is bounded by $O(max(n,m))$. Where $n = |M_1|$ and $m = |M_2|$. Thus the complexity of this step is $O(|V| * max(n,m)) = O(m * n * max(n,m))$.

Algorithm Complexity

Here we summarize the overall complexity of the algorithm. Let $k = max(|M_1|, |M_2|)$.

The first step of the algorithm takes $O(k^3)$, the second step takes $O(k^4)$ and the last, clustering step, takes $O(k^3)$. Thus the overall complexity is bounded by $O(k^4)$.

We have implemented the FlexProt algorithm in C++ and performed extensive experimentation (see section) on an Intel based personal computer (Pentium II 400MHz with 256 Mb RAM under the Linux operating system). The implementation of the *"Single-source shortest paths"* algorithm was taken from the LEDA library (Mehlhorn 1999). The running times ranged from 3.75 sec. for molecules of 150 amino acids to 41.9 sec. for molecules of 700 amino acids. The average running time was approximately 7 sec. See table (1) for a representative list of results and Appendix B of for the parameters used.

Experimental Results

We have conducted numerous experiments with FlexProt algorithm. Initially, we tested the algorithm mainly on examples from Gerstein's database of macromolecular motions (Gerstein & Krebs 1998). In this database the proteins are classified hierarchically into a limited number of categories on the basis of size - fragment, domain, and subunit motions. Each category is classified according to the type of the flexible motion - hinge, shear and "other" . Gerstein's database includes mainly protein pairs which represent different conformations of the same protein. These examples do not demonstrate adequately the full power of our algorithm, since the residue correspondence problem there is trivial (each residue corresponds to itself in the aligned protein). The power of FlexProt is in his ability to solve simultaneously both the residue correspondence and hinge region detection tasks in proteins, We must emphasize, however, that we never exploited the "known" correspondence of the Gerstein database protein pairs. The input to the algorithm was standard, thus no *a-priori* correspondence knowledge was assumed.

Nevertheless, to test the algorithm ability to align non similar protein structures, we have conducted larger scale experiments on the flexible molecules using the SCOP (Murzin *et al.* 1995) database. We made several comparisons of protein pairs belonging to different levels of the SCOP tree. As one ascends the SCOP levels the sequences homology and the structural similarity diminishes.

Some results are summarized in Table 1. The figures of these experiments were prepared using RasMol software (Sayle & Milner-White 1995).

All the experiments were conducted on a 400 MHz Pentium©II processor with 256MB internal memory with a Linux operating system.

The Data Set

We present four sets of experiments. In the first three the motions are classified by (Gerstein & Krebs 1998) as "hinge", while in the last one as "other".

Glutamine Binding Protein. We compared the glutamine binding protein (**1ggg**, chain A) in open, ligand-free, form with the complex when it is bounded to glutamine (**1wdn**, chain A). The program detected two hinge conformations of one structure with respect to the other. The hinges are located at residues 87-88 and 180-181. The RMSD of the total matching set is 0.96. Additionally we took from the SCOP database a histidine binding protein complexed with histidine (**1hpb**). The histidine binding protein and glutamine binding protein according to SCOP belong to the same family "Phosphate binding protein-like" proteins. We compared **1ggg** (chain A) with **1hpb**. The program found 4 similar fragments, two of them had similar transformations but were separated by a turn located at residue 132-135 of the **1hpb**, resulting in 3 matched clusters with total RMSD 2.07. For details see figure (3) and table (1).

Motion in Calmodulin. Calmodulin (CaM) is a C_{α}^{2+} binding protein. It is involved in a wide range of cellular C_{α}^{2+} -dependent signaling pathways. It regulates the activity of a large number of proteins including protein kinases, protein phosphatases, nitric oxide synthase, inositol triphosphate kinase, nicotinamide adenine dinucleotide kinase, cyclic nucleotide phosphodiesterase, C_{α}^{2+} pumps, and proteins involved in motility (Ikura *et al.* 1992). We compared human calmodulin (**1cll**) with calmodulin complexed with rabbit skeletal myosin light-chain kinase (**2bbm**, chain A). The program detected a hinge motion in the α-helix connecting the two domains. The hinge is located at the region of residues 78,79. We also compared the complexed Calmodulin (**2bbm**, chain A) with Troponin C protein (**1top**) which has a similar structure to Calmodulin. In SCOP it is classified as a Calmodulin-like protein.

Table 1: ***FLexProt*** Experimental Results†

Protein Pair	Backbone Length	Number of Flexible Regions	Match List Size	Matched Rigid Fragments	Total RMSD	Time (sec)
2bbm(chain A) 1cll	148 144	1	144	(4...78)-(79...147) (4...78)-(79...147)	2.22	3.75
2bbm(chain A) 1top	148 162	3	147	(2...25)-(26...63)-(64...76)-(77...148) (12...35)-(36...73)-(76...88)-(90...161)	2.43	4.48
2ak3(chain A) 1ake(chain A)	226 214	2	205	(6...120)-(121...164)-**[(166...194)-(195...211)]** (1...115)-(118...161)-**[(167...195)-(198...214)]**	2.53	6.9
2ak3(chain A) 1uke	226 193	1	184	**[(1...106)-(107-116)]-[(117...127)-(158...192)-(193..214)]** **[(2...107)-(111-120)]-[(121...131)-(136...170)-(172..193)]**	2.31	5.09
1bpd 2bpg(chain A)	324 324	1	324	(9...88)-(89...335) (9...88)-(89...335)	1.81	13.37
1dpe 1dpp(chain A)	507 507	2	507	(1...262)-(263...480)-(481...507) (1...262)-(263...480)-(481...507)	0.58	25.89
1ggg(chain A) 1wdn(chain A)	220 223	2	220	(5...87)-(88...180)-(181..224) (5...87)-(88...180)-(181..224)	0.96	7.25
1ggg(chain A) 1hpb	220 239	2	220	(5...89)-**[(90...130)-(131...181)]**-(182...224) (7...91)-**[(92...132)-(135...185)]**-(192...234)	2.07	7.46
1ncx 1tnw(Model 1)	162 162	3	161	(1...35)-(36...68)-(69...92)-(93...161) (1...35)-(36...68)-(69...92)-(93...161)	2.7	4.93
1mcp (chain L) 4fab (chain L)	220 219	1	218	(2...110)-(111...219) (1...109)-(110...218)	1.93	7.92
1mcp (chain L) 1tcr (chain B)	220 236	1	213	**[(1...29)-(37...56)-(57...115)]-[(116...205)-(206...220)]** **[(1...29)-(30...49)-(54...119)]-[(123...216)-(232...246)]**	2.4	9.5
1lst 2lao	239 238	2	238	(1...90)-(91...177)-(178...238) (1...90)-(91...177)-(178...238)	1.35	8.30
1lfh 1lfg	691 691	2	691	(1...84)-(85...244)-(245...691) (1...84)-(85...244)-(245...691)	1.41	41.90
1ddt 1mdt(chain A)	523 523	1	523	(1...392)-(393...535) (1...392)-(393...535)	1.58	32.84
3gap(chain A) 3gap(chain B)	208 205	1	205	(1...130)-(131...205) (1...130)-(131...205)	1.8	6.79

†The first column gives the PDB file names of matched molecules. The second column indicates the size of each molecule. The third column lists the number of flexible regions found between the compared molecules. The fourth column gives the total number of matched C_α atoms. The fifth column lists the matched consecutive fragments. The group of fragments typed in bold and enclosed in square brackets represents clusters of fragments having the same 3-D transformation. The sixth column lists the RMSD of the total matching set. The last column gives the running time of the program.

```
6      1top          28   29                           59   60               78   79
 HHHHHHHHHHHHH SSSS  B ~~~HHHHHHHHHHHH    SHHHHHHHHTT SS ~~~SSSB HHHHHHHHS     ~~~S  THHHHHHHHHHS SSSS
EEQIAEFKEAFSLFDKDGDGTIT~~~TKELGTVMRSLGQNPTEAELQDMINEVDADG~~~NGTIDFPEFLTMMARKMKD~~~TDSEEEIREAFRVFDKDGNG
EEMIAEFKAAFDMFDADGGGDIS~~~TKELGTVMRMLGQNPTKEELDAIIEEVDEDG~~~SGTIDFEEFLVMMVRQMKE~~~GKSEEELANCFRIFDKNADG
HHHHHHHHHHHHTTT S SSEEE~~~GGGHHHHHHHTT    HHHHHHHHHHH TTS~~~  EEEHHHHHHHHHHHHHH~~~HHHHHHHHHHHHHH TT SS
16     2bbm          38   39                           69   70               88   92

       1top                                148
SB HHHHHHHHHHTTS   HHHHHHHHHHH SSS SSB HHHHHHHH
FISAAELRHVMTNLGEKLTDEEVDEMIREADIDGDGQVNYEEFVTMMTSK
FIDIEELGEILRATGEHVTEEDIEDLMKDSDKNNDGRIDFDEFLKMMEGV
 B HHHHHHHHHHTTT   HHHHHHHHHHH TT SSSB HHHHHHHHHS
       2bbm                                161
```

Figure 1: Calmodulin (**2bbm**, chain A) compared with Troponin C (**1top**). The conformation is composed of three hinges: 28-29 (38-39), 59-60 (69-70), 78-79(88-92), residues of the **1top** (**2bbm**).

```
6      2ak3
 EEEEE  TTSSHHHHHHHHHHHHB  EEEHHHHHHHHHHTT HHHHHHHHHHHTT    HHHHHHHHHHHHHHHTTS EEEES   SHHHHHHHHTT
LRAAIMGAPGSGKGTVSSRITKHFELKHLSSGDLLRDNMLRGTEIGVLAKTFIDQGKLIPDDVMTRLVLHELKNLTQYNWLLDGFPRTLPQAEALDRAYQID
MRIILLGAPGAGKGTQAQFIMEKYGIPQISTGDMLRAAVKSGSELGKQAKDIMDAGKLVTDELVIALVKERIAQEDCRNGFLLDGFPRTIPQADAMKEAGIN
 EEEEEESTTSSHHHHHHHHHHHH   EEEHHHHHHHHHHHT TTTGGGHHHHHHT    HHHHHHHHHHHHHSGGGGG EEEES   SHHHHHHHHHTT
1      1ake

          120   121        2ak3                          164   166                        194   195
EEEEEE  HHHHH~~~HHHT EEEETTTTEEEETTTB  SSTTB TTT  B B  GGGSH~~~HHHHHHHHHHHHHHHHHHHHHHHT EEEE~~~E SSHHH
TVINLNVPFEVIK~~~QRLTARWIHPGSGRVYNIEFNPPKTMGIDDLTGEPLVQREDDRP~~~TVVKRLKAYEAQTEPVLEYYRKKGVLETF~~~SGTETNK
VDYVLEFDVPDEL~~~DRIVGRRVHAPSGRVYHVKFNPPKVEGKDDVTGEELTTRKDDQE~~~RLVEYHQMTAPLIGYYSKEAEAGNTKYAK~~~GTKPVAE
 SEEEEEE  GGG~~~HHHHTEEEETTTTEEEETTTB  SSTTB TTT  B B  TT SH~~~HHHHHHHTTTHHHHHHHHHHHTTSSEEEE~~~TTS HHH
          115   118        1ake                          161   167                        195   198

2ak3   211
HHHHHHHHHH
IWPHVYAFLQ
VRADLEKILG
HHHHHHHHH
1ake   214
```

Figure 2: Adenylate Kinase (**2ak3**, chain A) compared with adenylate kinase complexed with the inhibitor AP=5=A (**1ake**, chain A).The first flexible region is located between residue 120 and 121 (115 and 118) of the **2ak3** (**1ake**) , the second flexible region is located between residue 164 and 166 (161 and 167) of the **2ak3** (**1ake**). The last two fragments were joined into one cluster since they shared similar transformation but were separated by the short flexible region, residues 164-166 (161-167) of the **2ak3** (**1ake**).

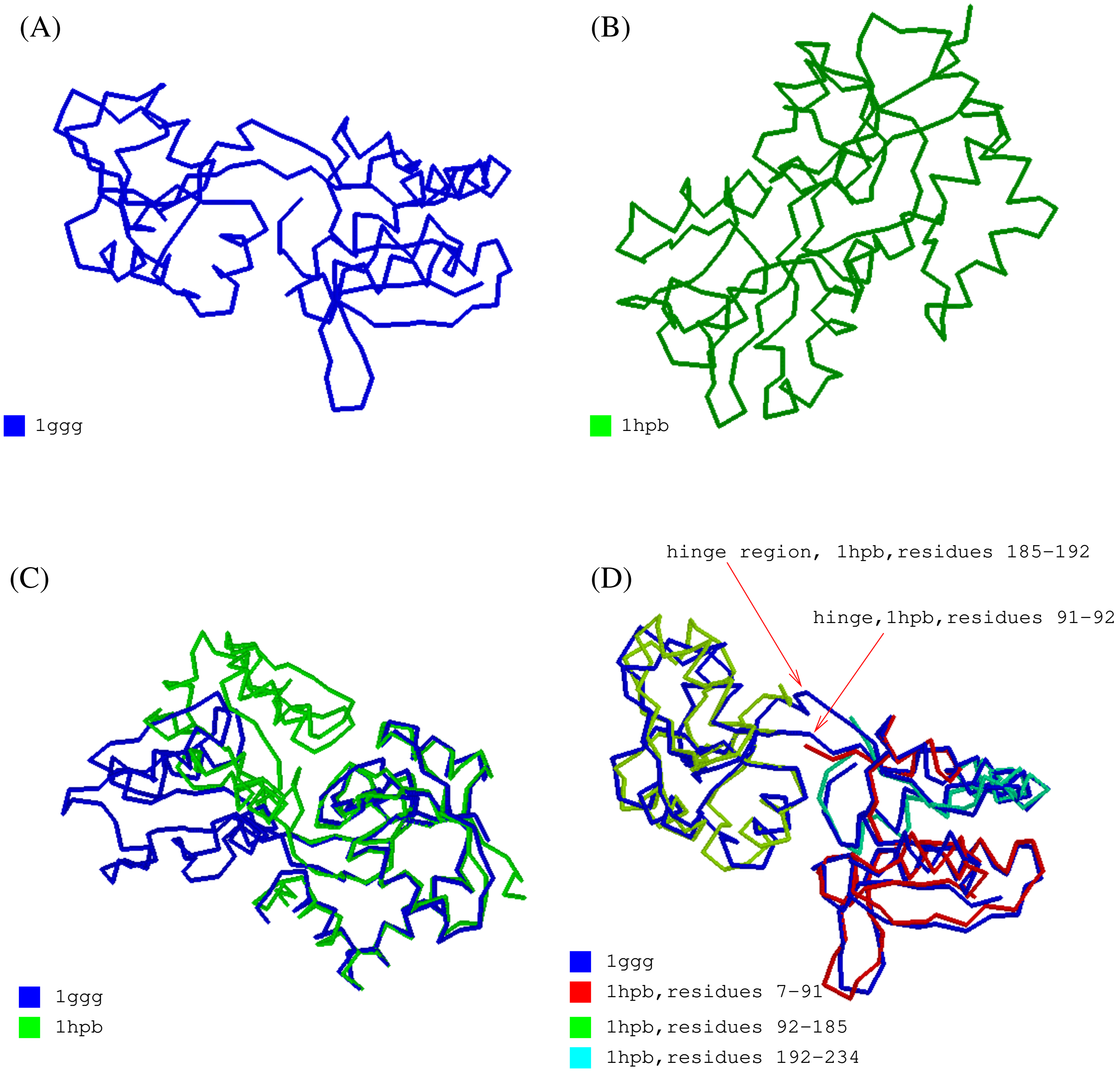

Figure 3: Glutamine binding protein **1ggg** (chain A) compared with histidine binding protein **1hpb**. Pictures (a), (b) show structure of **1ggg**, **1hpb** respectively. Picture (c) displays the best rigid superimposition. Notice that on the left side of the picture the protein chains are not aligned. Picture (d) shows superimposition of **1hpb** on **1ggg** with respect to the flexible regions found by the program. The resulting structural alignment is almost complete.

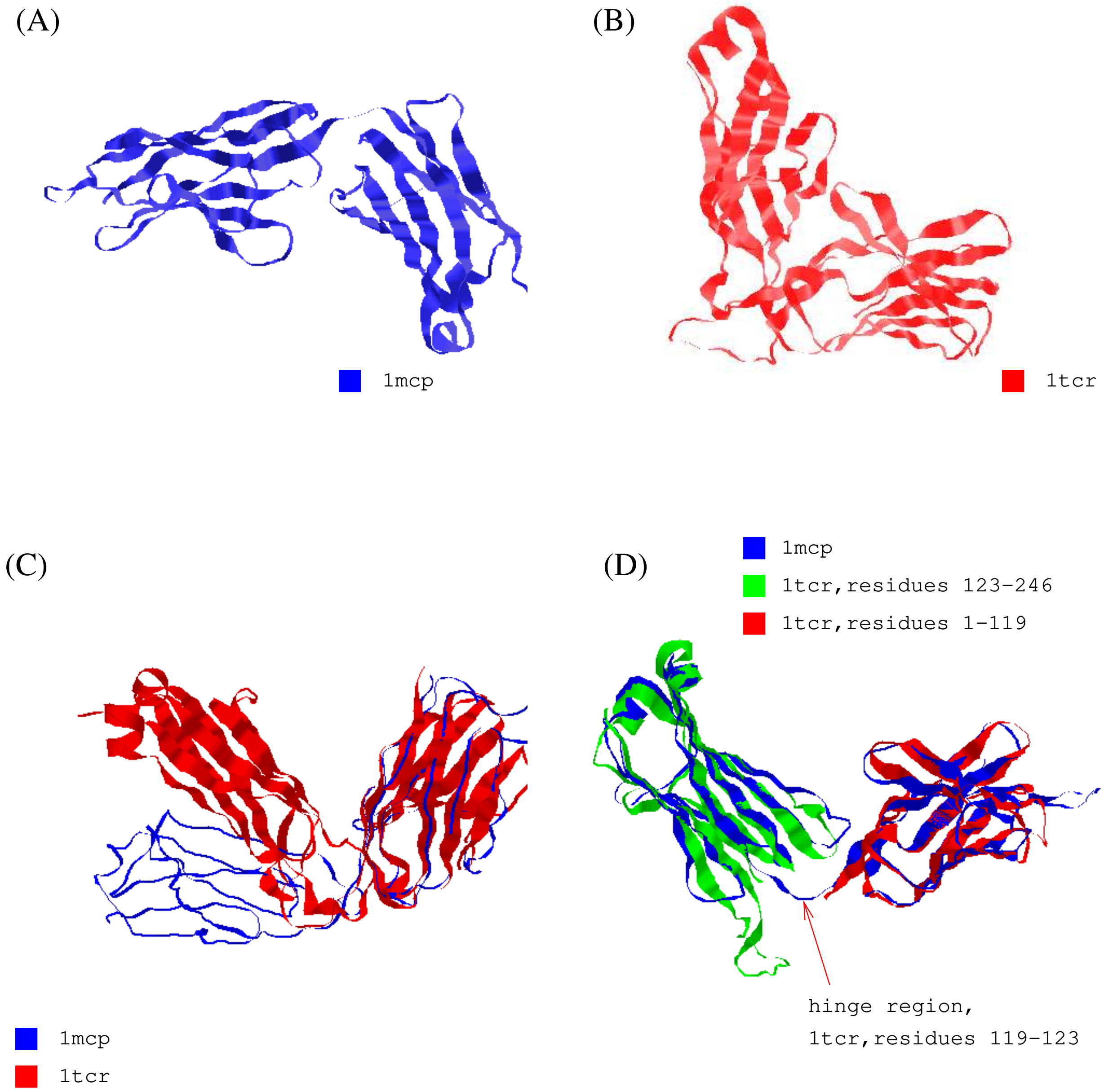

Figure 4: Immunoglobulin fab fragment **1mcp**(chain L) is matched with murine T-cell antigen receptor **1tcr** (chain B). Pictures (a), (b) show a ribbon representation of the structures of **1mcp**(chain L), **1tcr**(chain B) respectively. Picture (c) shows the best rigid superimposition between the two molecules. Picture (d) shows hinge region detected by the program, **1tcr** is superimposed on to **1mcp**.

Indeed the program detected 4 similar rigid fragments separated by 3 hinge regions. See the alignment in 1.

Adenylate Kinase. We compared Adenylate Kinase isoenzyme-3 (**2ak3**, chain A) with Adenylate Kinase complexed with the inhibitor AP=5=A (**1ake**, chain A). The program found 4 structurally similar regions. The last two had similar transformations and had been clustered to one group. The first flexible region is between residue 120 and 121 (115 and 118) of the **2ak3** (**1ake**), the second flexible region is between residues 164 and 166 (161 and 167) of the **2ak3** (**1ake**). See the alignment in figure 2.

From the SCOP database we took UMP/CMP kinase (**1uke**). Due to SCOP classification the UMP/CMP kinase and Adenylate kinase belong to the family of "Nucleotide and nucleoside kinases". We compared **2ak3** with the **1uke**. The program found 5 structurally similar regions. The first two were separated by a small loop (residues 106-107 of **2ak3**), but shared the same transformation, so they were clustered into one rigid matching. The next 3 regions also had a similar transformation. The two α-helices shared the same transformation but were separated by a large fragment found in **2ak3** (residues 127-158) but not in **1uke**, where there is only a small loop connecting the two α-helices. Finally, these 3 regions were clustered into one matching set. The program detected a total of 2 matching clusters with RMSD 2.31 (for details see table 1). For this particular case we changed the program parameter RMSD threshold to 2.5 Angstrom, while the default was 3 Angstrom.

Immunoglobulin (Fab Elbow Joint) We tested our program on a "Fab Elbow Joint-like" motion (Lesk & Chothia 1988). We compared immunoglobulin Fab fragment taken from **1mcp** (chain L) versus one from **4fab** (chain L). Each chain is composed of two domains connected by an extended strand. The program detected the flexible region at residues 109-111 with total RMSD of 1.93. We also took from the SCOP database a murine T-cell antigen receptor (**1tcr**, chain B), which, according to SCOP classification, together with **1mcp** belongs to a "V set domains (antibody variable domain-like)" family. The program found two clusters which represented two domains separated by a flexible part (see figure at 4), with total RMSD of 2.4.

Conclusions and Future Work

We have presented a powerful tool for automatically matching protein molecules, enabling domain, or subdomain motions without *a priori* knowledge of the flexible regions between the domains, subdomains or other structurally preserved fragments.

Our tool can be utilized for extensive comparisons of proteins and other macromolecules in search for common domains and flexible joins.

The presented algorithm succeeded to resolve a large set of examples. Our method is very efficient. For average size proteins, consisting of about 220 residues, it takes approximately 7 seconds on a standard desktop PC. We intend to further optimize the performance of our method and apply it on large databases.

Acknowledgments

We thank Meir Fuchs and Ram Nathaniel for valuable discussions and contribution of software to this project. The research of H. J. Wolfson and R. Nussinov in Israel has been supported in part by the *Israel Science Foundation* administered by the *Israel Academy of Sciences* - Center of Excellence Program, by the Israeli Ministry of Science infrastructure grant, and by a Magnet grant from the Chief Scientist of the Ministry of Industry and Trade. The research of H.J.W. is partially supported by the Hermann Minkowski-Minerva Center for Geometry at Tel Aviv University. The research of R.N. has been supported in part with Federal funds from the National Cancer Institute, National Institutes of Health, under contract number NO1-CO-56000. The content of this publication does not necessarily reflect the view or policies of the Department of Health and Human Services, nor does mention of trade names, commercial products, or organization imply endorsement by the U.S. Government.

Appendix A: Calculation of the rigid transformation and RMSD

Suppose we are given a matching set of 3-D points $\{u_i\}_1^n$ and $\{v_i\}_1^n$. We need to find the rigid transformation T that minimizes:

$$\Delta = \min_T \sum_{i=1}^{n} |Tu_i - v_i|^2. \tag{1}$$

(RMSD is defined by $RMSD = \sqrt{\frac{\Delta}{n}}$)

We represent T as $Tu_i = Ru_i + a$, where $\mathbf{R}$ is rotation matrix and $\mathbf{a}$ is translation vector. Following the exposition in (Schwartz & Sharir 1987) one can show that the translation $\mathbf{a}$ is actually the translation between the centroids of the point sets $\{u_i\}_1^n$ and $\{v_i\}_1^n$, the 3×3 rotation matrix $\mathbf{R}$ is the one which maximizes

$$\delta = \max_R \sum_{i=1}^{n} Ru_i * v_i$$

and can be analytically computed in $O(n)$ time, and the minimum Δ is given by

$$\begin{aligned} \Delta \;=\; & \sum_{i=1}^{n} |v_i|^2 - \frac{1}{n} |\sum_{i=1}^{n} v_i|^2 \\ & + \sum_{i=1}^{n} |u_i|^2 \\ & - 2tr((A^T A)^{1/2}) \end{aligned} \tag{2}$$

Where the 3×3 matrix A is defined by

$$\forall i,j = 1,2,3 \quad A_{ij} = \sum_{k=1}^{n} u_k^i v_k^j \tag{3}$$

where $u_k = (u_k^1, u_k^2, u_k^3)$.

In section "Detection of Congruent Rigid Fragment Pairs" when we extend an initial match-list by adding one atom pair we have to recalculate Δ for the new match-list . One can easily observe from equation (2) that this can be done in $O(1)$ time. Thus, the whole process takes $O(n)$ where $\mathbf{n}$ is the size of the resulting match-list. At the end of the process we can calculate the translation vector $\mathbf{a}$ and the rotation matrix $\mathbf{R}$ in $O(1)$ time.

Appendix B: Program Parameters

For most cases the default parameter values were sufficient. Only for one experimental result (Adenylate Kinase, 2ak3 chain A versus 1ake chain A) we changed the RMSD threshold from 3 Angstroms to 2.5 Angstroms.

Model -	The name of first molecule.
ModelChain -	The chain ID of the first molecule (if needed).
Target -	The name of second molecule.
TargetChain -	The chain ID of the second molecule (if needed).
IJshift -	Maximum allowed shifting between two fragments, i.e. for any $F_i^1 F_j^2(l)$ $\|i-j\| \leq IJshift$. Default 50.
MaxDels -	Maximum allowed deletions during the fragments construction. Default 50.
MaxIns -	Maximum allowed insertions during the fragments construction. Default 50.
MinFragSize -	The minimum size of rigid fragment. Default 10.
RMSD -	The RMSD threshold for the matching set of rigid fragments. Default 3 Angstroms.
MaxNumFlex -	The maximum number of rigid fragments in the solution. Default 20.
ResNum -	The number of output results. Default 10.

References

Branden, C., and Tooze, J. 1991. *Introduction to Protein Structure*. New York and London: Garland Publishing, Inc.

Cormen, T.; Leiserson, C.; and Rivest, R. 1990. *Introduction to Algorithms*. MIT Press. chapter 25.4.

Gerstein, M., and Krebs, W. 1998. A database of macromolecular motions. *Nucleic Acids Research* 26(18). http://bioinfo.mbb.yale.edu/MolMovDB/.

Gerstein, M.; Lesk, A.; and Chothia, C. 1994. Structural Mechanisms for Domain Movements in Proteins. *Biochemistry* 33(22):6739–6749.

Gusfield, D. 1997. *Algorithms on Strings, Trees, and Sequences : Computer Science and Computational Biology*. Cambridge University Press.

Ikura, M.; Clore, G.; Gronenborn, A.; Zhu, G.; Klee, C.; and Bax, A. 1992. Solution structure of a calmodulin-target peptide complex by multidimensional NMR. *Science* 256:632–644.

Lemmen, C., and Lengauer, T. 2000. Computational Methods for the Structural Alignment of Molecules. *J. of Computer-Aided Mol. Design* 14:215–232.

Lesk, A., and Chothia, C. 1988. Elbow motion in the immunoglobulins involves a molecular ball and socket joint. *Nature* 335:188–190.

Mehlhorn, K. 1999. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press.

Mitchel, E.; Artymiuk, P.; Rice, D.; and Willet, P. 1989. Use of Techniques Derived from Graph Theory to Compare Secondary Structure Motifs in Proteins. *J. Mol. Biol.* 212:151–166.

Murzin, A.; Brenner, S.; Hubbard, T.; and Chothia, C. 1995. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.* 247:536–540.

Nussinov, R., and Wolfson, H. 1991. Efficient Detection of three-dimensional strcutral motifs in biological macromolecules by computer vision techniques. *Proc. Natl. Acad. Sci. USA* 88:10495–10499. Biophysics.

Rarey, M.; Kramer, B.; Lengauer, T.; and Klebe, G. 1996. A fast flexible docking method using incremental construction algorithm. *J. Mol. Biol.* 261:470–489.

Rigoutsos, I.; Platt, D.; and Califano, A. 1996. Flexible 3D-Substructure Matching and Novel Conformer Derivation in Very Large Databases of 3D-Molecular Information. Technical report, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY.

Sandak, B.; Nussinov, R.; and Wolfson, H. 1995. An Automated Robotics-Based Technique for Biomolecular Docking and Matching allowing Hinge-Bending Motion. *Computer Applications in the Biosciences (CABIOS)* 11:87–99.

Sayle, R., and Milner-White, E. 1995. Rasmol: biomolecular graphics for all. *Trends Biochem. Sci.* 20(9):374.

Schwartz, J., and Sharir, M. 1987. Identification of Partially Obscured Objects in Two Dimensions by Matching of Noisy 'Characteristic Curves'. *The Int. J. of Robotics Research* 6(2):29–44.

Shatsky, M.; Fligelman, Z.; Nussinov, R.; and Wolfson, H. 2000. Alignment of Flexible Protein Structures. In *8th International Conference on Intelligent Systems for Molecular Biology*, 329–343. The AAAI press.

Taylor, W. R., and Orengo, C. A. 1989. Protein structure alignment. *J. Mol. Biol.* 208:1–22.

Verbitsky, G.; Nussinov, R.; and Wolfson, H.

1999. Structural Comparison Allowing Hinge Bending, Swiveling Motions. *PROTEINS: Structure, Function and Genetics* 34:232–254.

Vriend, G., and Sander, C. 1991. Detection of Common Three-Dimensional Substructures in Proteins. *Proteins* 11:52–58.

Wolfson, H. J. 1991. Generalizing the Generalized Hough Transform. *Pattern Recognition Letters* 12(9):565 – 573.