Journal of Computer-Aided Molecular Design, 10 (1996) 41–54
ESCOM

J-CAMD 318

Placement of medium-sized molecular fragments into active sites of proteins

Matthias Rarey*, Stephan Wefing and Thomas Lengauer

German National Research Center for Information Technology (GMD), Institute for Algorithms and Scientific Computing (SCAI), Schloss Birlinghoven, D-53754 Sankt Augustin, Germany

Received 5 June 1995
Accepted 8 September 1995

Keywords: Molecular docking; Flexible docking; Receptor–ligand interaction; Molecular flexibility; Conformational analysis; Drug design

Summary

We present an algorithm for placing molecular fragments into the active site of a receptor. A molecular fragment is defined as a connected part of a molecule containing only complete ring systems. The algorithm is part of a docking tool, called FLEXX, which is currently under development at GMD. The overall goal is to provide means of automatically computing low-energy conformations of the ligand within the active site, with an accuracy approaching the limitations of experimental methods for resolving molecular structures and within a run time that allows for docking large sets of ligands. The methods by which we plan to achieve this goal are the explicit exploitation of molecular flexibility of the ligand and the incorporation of physicochemical properties of the molecules. The algorithm for fragment placement, which is the topic of this paper, is based on pattern recognition techniques and is able to predict a small set of possible positions of a molecular fragment with low flexibility within seconds on a workstation. In most cases, a placement with rms deviation below 1.0 Å with respect to the X-ray structure is found among the 10 highest ranking solutions, assuming that the receptor is given in the bound conformation.

Introduction

Since the first structures of receptors have become available through X-ray crystallography, scientists have been interested in automatically predicting the binding strength and the configuration in which small molecules like substrates, transmitters, or inhibitors bind to their receptors. A computer program yielding this prediction can be of significant help in constructing, finding and optimizing lead structures in the drug design process.

The main reason why the docking problem is hard to solve is that it is a many-body problem, coupled with a complex cost function. Because we are only interested in approximate solutions of the problem, we introduce the following simplifications and assumptions:

(1) The number of degrees of freedom is drastically reduced to the torsional degrees of freedom of the ligand (including conformational changes in flexible ring systems) and the six degrees of freedom of its overall translation/rotation. All receptor atoms are held fixed in space.

(2) Only the receptor–ligand complex is analyzed; neither of its two components is explicitly considered by itself (no closed thermodynamic cycle).

(3) A configuration is assumed to exist that strongly dominates the partition function of the complex and thus the strength of binding. In this 'binding mode', the receptor and the ligand are supposed to take on complementary shapes.

Simplifications 1 and 2 are assumed to be compensated for by a modified potential, whereas assumption 3 permits us to regard the calculation of a binding constant as an optimization problem. Nevertheless, this optimization problem is not easy to solve because, e.g., comparing two-dimensional patterns on flexible surfaces of molecules is a complicated task for a computer.

While the earlier algorithms simplify the docking problem by regarding both molecules as rigid [1–6], more recent approaches try to model molecular flexibility directly. If time efficiency is needed, integrating molecular flexibility of the ligand in automated docking tools is

*To whom correspondence should be addressed.

mostly done by cutting the ligand into fragments in such a way that, in a discrete model, the number of energetically favorable conformations for each fragment is bounded by a constant. With this concept in mind, there are two different strategies for docking: (i) place and combine, i.e., placing the fragments independently and linking the solutions to form placements for the whole ligand; and (ii) incremental build-up, i.e., placing one fragment as a base or anchor fragment and building up the ligand incrementally from this fragment. Both approaches have been successfully applied in automated docking and de novo design tools. Examples of tools using the place and combine strategy are extended-DOCK [7], HOOK [8], LEGO [9], the hinge-bending algorithm [10], and the first version of LUDI [11]. Examples of the incremental build-up strategy are the (peptide) docking tool GROW [12], the current version of LUDI [13], SPROUT [14,15], GenStar [16], and the backtracking algorithm of Leach and Kuntz [17]. A review of site-directed structure generation can be found in Ref. 18.

Some approaches not based on fragmentation should also be mentioned. Goodsell and Olson [19] have presented a flexible docking algorithm based on simulated annealing. The disadvantage of this approach is that the quality of the results depends highly on a user-defined start configuration. A new approach in molecular docking is to enumerate all interaction patterns in all conformations of the ligand and place the ligand into the receptor when the pattern is found [20]. Of course, the need for selecting a base fragment is avoided, but the combinatorial complexity is not reduced. The run time of this method increases exponentially with the number of rotatable bonds and the method can therefore only be used for docking ligands with a limited number of conformations.

Most of the tools mentioned above contain automatic methods for placing molecular fragments. Before we start explaining our approach, we want to clarify the differences and the advantages of our algorithm compared to this related work. In the DOCK algorithm [1,7], spheres describing the negative image of the pocket are used for matching ligand atoms. The algorithm places the ligand such that a good complementary fit between the surfaces is achieved. This is a suitable approach as long as the ligands are large and not very flexible, because steric fit is a necessity for developing weak chemical interactions between the molecules, and geometry can be used as a time-efficient and effective filter. When smaller ligands or fragments are considered, filtering by geometry alone becomes ineffective because a small ligand can be placed nearly everywhere in the pocket with significant contact area between the molecules. In our docking algorithm, chemical interactions are the basis for placing the fragments. Thus, we avoid the generation of a large set of geometrically feasible solutions that later has to be reduced by checking physicochemical properties of the complexes.

This direct modeling of physicochemical properties and the usage of this information for computing placements is the common part of our algorithm and the de novo design tool LUDI [13]. In fact, the model of molecular interactions as well as the energy estimation function were taken from LUDI and only slightly changed. LUDI constructs five ligand–receptor interactions simultaneously and computes a transformation to achieve these interactions by a superposition algorithm. This procedure would be expensive if all possible matches of five interacting groups are considered. Thus, LUDI contains heuristic rules to reduce the number of superpositions.

The main focus of this paper is a new algorithmic engine for the matching problem in molecular docking. As proposed by Fischer et al. [21], hashing techniques for triangles are used. We modify their approach in two aspects. First, since we are modeling chemical interactions explicitly, our point sets are larger. The geometric hashing technique used in Ref. 21 has cubic space requirements, which is unacceptably large for our purposes. Our data structure needs only quadratic space in the number of matching points. Second, in geometric hashing, clustering is done on the basis of the triangles. This choice gives the approach a heuristic flavor. We avoid this in our approach by computing a transformation first and developing a time-efficient algorithm to explicitly cluster the placed molecules.

The presented algorithm is part of our docking program FLEXX [22]. FLEXX is based on the incremental build-up strategy and the algorithm is successfully used for placing base fragments into the active site in only seconds of CPU time per conformation on a workstation. In most cases, a placement similar to the X-ray structure of the complex is found among the highest ranking solutions.

In the next section, the docking problem is described and modeled in a more formal way. Subsequently, our algorithm for placing molecular fragments into the active site of proteins is presented. Finally, we give some results obtained with FLEXX by trying to reproduce known receptor–ligand complexes.

Methods

Modeling the docking process

Input data

The input data of the docking problem consist of an appropriate description of the receptor, including the location of the active site, and the ligand. The assumption that the active site is known is only a weak restriction, because our computer program FLEXX is part of a software package that combines suitable biochemical data-

bases with molecular modeling and flexible docking. Furthermore, methods to predict the location of the active site have been reported in Refs. 1, 23 and 24. Therefore, we can expect knowledge about the active site to be available in practical cases.

The physicochemical properties of the receptor, which are relevant for the docking process, are presented as labels of the atoms. These are: the chemical element, the number and position of bound hydrogens, hybridization, partial charges and van der Waals radii of the atoms, possible interactions and interaction geometries. Information not available in the standard PDB format [25] is defined by the user of FLEXX in a special input file, called the receptor description file. This file contains a set of rules describing how to map amino acid templates to the loaded amino acids, how to solve ambiguities in the PDB file, and how to add hydrogens to polar atoms. In addition, the active site is described.

The ligand is described by its molecular graph, consisting of a set of atoms (the vertices) and bonds between them (the edges). Each atom is labeled with a set of physicochemical properties similar to those of the receptor. Additional information needed for the computation of low-energy conformations is attached to the graph. In FLEXX, a discrete model of conformational flexibility is used. More precisely, bond lengths, bond angles and torsional angles at double or triple bonds are held fixed, while suitable torsional angles of single bonds and conformations of ring systems are represented as discrete sets. This model of discrete conformational flexibility is nearly the same as that used in the conformational search program MIMUMBA [26].

We have tested this model for a set of different inhibitors by comparing the ligand conformation in the complex determined by X-ray crystallography with the most similar conformation, which can be found in the conformational model of MIMUMBA (bond lengths and angles are taken from an energy-minimized structure). In all cases, the rms deviation is below 0.5 Å, which is sufficiently accurate for our purposes.

Molecular interactions

For the positioning of the fragment into the active site, only localized interactions are modeled (e.g., hydrogen bonds, salt bridges and special hydrophobic interactions (phenyl ring–phenyl ring, phenyl ring–methyl group, phenyl ring–amide group)). An interaction is represented by an interaction center and parts of a spherical surface around this center, called the interaction surface (see Fig. 1). An interaction between two groups appears if the interaction center of the first group lies on the interaction surface of the second group and vice versa (see Fig. 2). With this model, the geometric rules defined for hydrogen bonds and specific hydrophobic interactions, as revealed by analysis of crystallographic data [27–30], can be represented satisfactorily. For the purpose of fragment placement, each interaction surface at the receptor is approximated by a discrete set of points, called the interaction points. Some examples of interacting groups, their types, and their geometric requirements are listed in Table 1.

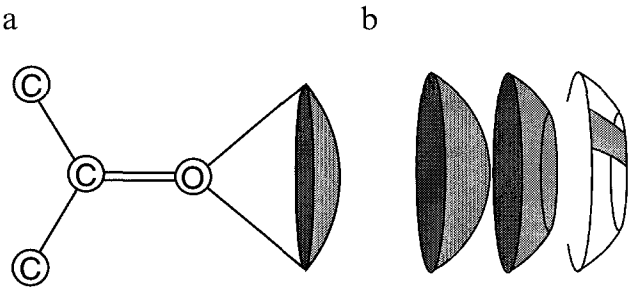

Fig. 1. Interaction surface of a carbonyl group (a) and different types of interaction surfaces in FLEXX: cone, cone section and spherical rectangle (b).

In order to estimate the free energy of binding, FLEXX uses a slight modification of the empirical free-energy function introduced by Böhm [31], which has been developed for the de novo design tool LUDI. This function includes terms for hydrogen bonds, salt bridges, hydrophobic contact area, and frozen degrees of freedom. The two main advantages of the function are that it can be evaluated efficiently and that the function is gauged against real binding constants. In FLEXX, the method for estimating the hydrophobic contact area has been slightly modified, but the differences between the results are sufficiently small to make a recalibration of the parameters unnecessary.

Static data

In FLEXX, physicochemical background knowledge is accumulated in a set of easily modifiable data files. The files contain information on:

(1) types of interaction and compatibilities between them, as well as basic parameters of the energy function;

(2) interaction geometries, consisting of an interaction center and a radius, angular constraints, and interaction-dependent parameters for the energy function derived from Refs. 13 and 27;

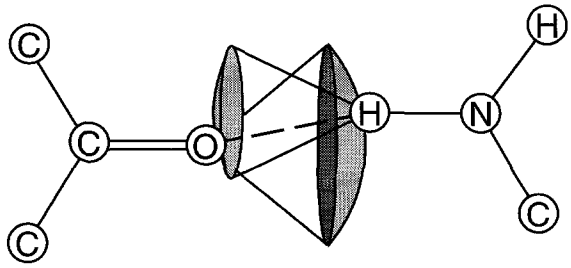

Fig. 2. Hydrogen bond between a carbonyl oxygen and a nitrogen. The interaction occurs if the interaction centers (oxygen and hydrogen atom) lie approximately on the interaction surface of the counter group.

TABLE 1
EXAMPLES OF INTERACTING GROUPS AND THEIR GEOMETRIC REQUIREMENTS

Type of interaction group	Example	Geometry	Center of sphere (section)
H-bond donors	O-H	Cone	H
	N-H	Cone	H
Metal ions	Zn^{2+}	Sphere	Zn
H-bond or metal-ion acceptors	C=O in COO^-	Two spherical rectangles	O
	C=O, S=O, V=O	Cone	O
	R-O-H	Two spherical rectangles	O
	R_2O, R_3N	Cone	O, N
Aromatic groups		Two cones above and below the plane	Center of the ring
Groups interacting with aromatic groups	Aromatic groups	Cone	H
	Methyl group	Sphere	C
	Amide group	Two cones above and below the plane	Middle of the C-N bond

(3) properties like hybridization, partial charge, number of bound hydrogens, hydrogen-bonding geometry, bond types and interactions on the atomic level in a set of templates for each amino acid;

(4) discrete sets of torsional angles for acyclic single bonds in the ligand derived from the MIMUMBA data file [26,32] (ring-system conformations are generated automatically by the program SCA [33]); and

(5) molecular fragments describing interacting groups in arbitrary organic molecules (ligands).

To map these static data onto the molecules, different strategies are used for the receptor and the ligand. The receptor consists of a small set of building blocks, the amino acids. In this case, a set of templates for each amino acid is defined. The assignment of a template to an amino acid is done by default rules, which can be overlaid with specific assignment rules by the user of the docking tool.

In contrast, the ligand is an arbitrary organic molecule that has no clearly defined building blocks above the atomic level. Here, physicochemical information is defined for small molecular fragments like carboxylate or carbonyl groups if hydrogen-bonding acceptors should be defined. The information is mapped onto the ligand by searching for predefined molecular fragments in the molecule. Although for this mapping no efficient algorithm is known in general, this step is not time-critical because the vertices and edges in the graph are labeled with information like the chemical element, the bond type, etc., and the fragments are relatively small.

The fragment-placing algorithm

The fragment-placing algorithm uses a method similar to a technique known in pattern recognition, called pose clustering [34]. The problem normally solved with this technique is the following. Given a 2D picture of a 3D scene and a 3D object, where both are sets of points, decide whether the object is part of the scene and compute the viewpoint of the camera producing the picture.

This problem is tackled with the following algorithm. Each triple of points of the object is mapped to each triple of points in the scene. From these matchings, a discrete set of possible viewpoints is computed. Subsequently, the viewpoints computed in the first phase are clustered. Each cluster that is larger than a given threshold represents a candidate for the object in the scene.

In the context of docking, the object is the set of interaction centers on the fragment. The scene is composed of the interaction points in the receptor pocket. Instead of viewpoints, we are looking for transformations of the fragment into the active site (see Fig. 3). The following differences make an adaptation of the algorithms for pose clustering to our problem necessary.

First of all, in our application the object and the scene have the same dimension. This means that the distances between two points in the object must be almost identical to the distances between their matching points in the scene. In addition, the points are labeled with the type of interaction they represent. To establish a match, these types must be compatible. Thus, there is only a small set of triples of points (interaction points) in the receptor that can successfully be matched to a given triple of points (interaction centers) in the ligand. Below in the section 'Searching for nearly congruent triangles', we develop a data structure that helps us to quickly retrieve these few triples. The angular requirements at the receptor interaction centers are fulfilled because the discrete interaction points are only generated on the interaction surfaces. The additional complementary angular requirements at the ligand interaction centers are checked after this retrieval.

Second, the fragment of the ligand has only a few dozen interaction centers and, for an energetically favorable placement, only a subset of these points must be matched. Thus, all clusters, and not only the large ones, are of interest. This makes heuristic strategies in the clustering procedure like those used in Ref. 35 inapplicable in our case. Instead, more exact clustering techniques must

be used. We describe such a technique below in the section 'Clustering transformations'.

At the end of the algorithm, a representative transformation is computed for each cluster and a clash test is performed. Optionally, the solutions can be optimized by energy-minimization techniques.

So far, the possible flexibility of the base fragment has not been considered. It can be integrated directly into the algorithm as follows. For each triangle constructed out of a triple of ligand interaction centers, the conformational set is divided into disjoint subsets, such that the triangles in all conformations of a subset are identical. After the retrieval of similar receptor triangles, a matching is generated for each conformation of the subset. The clustering at the end of the algorithm is done for each conformation independently.

Searching for nearly congruent triangles

In this section, the problem of finding triangles that are nearly congruent to a given triangle is discussed. Before explaining an efficient data structure for this problem, we want to clarify its application in solving the docking problem.

In the case of docking, the receptor is represented by a set of interaction points with different interaction types. For the ligand, a set of interaction centers is computed and we search for transformations of the ligand such that interaction centers are mapped onto interaction points. A transformation is uniquely defined by mapping three interaction centers of the ligand onto three interaction points of the receptor. A necessary condition for simultaneous occurrence of the three interactions is that the two triangles defined by the interaction centers of the ligand and the interaction points of the receptor have sides of nearly equal length. Thus, the problem can be stated independently from the specific application of molecular docking: given a set of points with types assigned to them in three-dimensional space and a triangle with types assigned to its corners, construct all triangles out of the given point set with the following properties: (i) the triangle side lengths are nearly equal to the side lengths of the given triangle; and (ii) the types at the triangle corners are compatible with the types assigned to the corners of the given triangle.

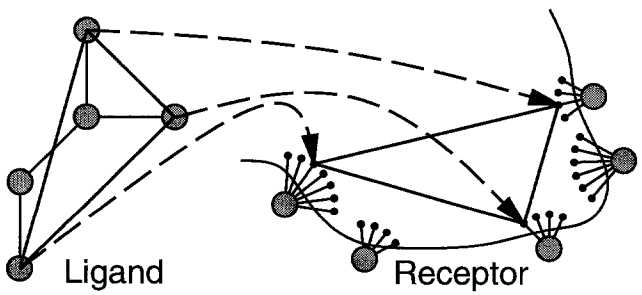

Fig. 3. The fragment-placing algorithm. Mapping three interaction centers (grey spheres) of the ligand onto three discrete interaction points in the active site (black dots) defines a unique transformation of the ligand into the active site.

A naive approach to this problem considers each possible triangle out of the point set and tests the required properties. This method requires no preprocessing time and no extra space, but the query time (run time to search matches for one triangle) is $O(m^3)$, where m is the number of points in the given set. This query time can be reduced significantly by using hashing techniques.

Hashing of triangles Hashing data structures for triangles are already in use in the areas of pattern recognition [36] as well as molecular docking [21]. Here we begin by describing a rough outline of a simple triangle hashing data structure, which should clarify the problems arising with this approach.

In the preprocessing step, each possible triangle that can be constructed out of the point set is considered. The number of different interaction types is small, thus we can construct a hashing table for each combination of three interaction types. The hash address of the triangle is computed on the basis of the side lengths of the triangle, rounded to integer values.

Given a triangle, the query proceeds as follows. First the side lengths of the triangle are rounded to integer values and the corresponding bucket in the hash table is determined. Since we are not looking for exact matches, we have to consider neighboring buckets as well. If the bucket width is set to twice the tolerance value for matching side lengths, we have to consider two buckets for each triangle side. Thus, eight hash addresses must be evaluated and the triangles found must be checked to see whether the differences in the side lengths are below the tolerance value.

This data structure has a very efficient query time. If we assume that the distances are distributed uniformly inside each pair of two adjacent buckets, the resulting query time is proportional to the number of retrieved triangles. On the other hand, the preprocessing time and the space needed increase dramatically to a value of the order m^3. Because the number of discrete interaction points in the receptor is normally quite large, this enormous space requirement is unacceptable. In practical cases, m is of the order 500–1000, which yields about 0.1 to 1 billion triangles. Even if the triangle edges are restricted to a reasonable interval, between 1 and 10 Å in molecular docking, this number of triangles is unacceptably high. In the following section a new data structure is presented, which uses $O(m^2)$ space and supports a query algorithm that in practice yields the same efficiency as the one described above.

Hashing of line segments Obviously, if we want to circumvent cubic space requirements, precomputing and storing every triangle is impossible. In this new approach, pairs of points defining line segments are stored in hash

tables. We can construct a set of hash tables in the same way as we have done for triangles.

In contrast to triangle hashing, we have to construct the triangles on-line during the query from selected line segments. This makes a more sophisticated query algorithm necessary, which we will outline now. For a given triangle, we are searching for line segments out of our point set matching two of the sides of the given triangle. This is done by applying the hashing technique for line segments. Two additional properties must be fulfilled by a pair of line segments so that an output triangle can be constructed from them. First, the line segments must have one end point in common and second, the length between the remaining two end points must be nearly equal to the length of the third side of the given triangle. Using list merging, we can achieve a situation where only pairs of line segments fulfilling the first property are considered. The second property is simply checked for all constructed triangles.

In conclusion, the following performance figures can be derived. The space requirement and the preprocessing time for the data structure are quadratic. For the query algorithm we cannot give a meaningful worst-case time bound, neither in terms of the length of the input nor in terms of the size of the output. However, tests have shown that the run time of the query algorithm is almost the same for triangle and line-segment hashing. A formal description of the data structure and the query algorithm is given in Appendix 1.

Using the data structure in molecular docking At the end of this section, we want to mention some additional features of the docking problem that are used in FLEXX in order to further improve the performance of our docking method. We can reduce the number of line segments that must be stored by applying two heuristics. First, in molecular docking only a small range of distances occurs. An intramolecular atom–atom distance is larger than a given threshold (about 1 Å) and a fragment of the ligand is not too large (about 10 Å for the maximum distance between two atoms). Thus, only line segments with lengths between these bounds are of interest and must be stored, even if the active site has a greater diameter. Second, the interactions considered in FLEXX are classified in two groups: active and less active. Hydrophobic interactions with a weak geometric specificity are characterized as being less active. In contrast, hydrogen bonds and salt bridges define restrictive geometric constraints and are thus characterized as being active. Each triangle matched must have at least two active interactions, otherwise the geometric constraints are too fuzzy to determine a transformation. Thus, line segments between less active interaction points need not be stored. Note that interaction patterns with more than one less active interaction can still be found because of the subsequent clustering step. However, interaction patterns with fewer than two interactions are not taken into consideration. In our benchmark set, we can always find three interactions. However, if FLEXX should run into a base fragment that is not able to form three interactions with the receptor, we offer an extension of our method that is able to work with only two interactions.

Clustering transformations

With the data structure developed in the previous section, we are now able to find all triangles of interaction points in the receptor that can be mapped on a given triple of interaction centers of the ligand. Each such assignment of two triangles defines a unique transformation of the ligand into the active site, such that the rms deviation between the two triples of end points is minimal. Of course, normally there are more than three interactions between ligand and receptor. This makes a clustering of the resulting transformations necessary.

The distance measure used in FLEXX for clustering the transformations is the rms deviation between different placements of the fragment. As shown in Appendix 2, this rms deviation can be computed in constant time after linear-time preprocessing.

The hierarchical clustering algorithm We now want to explain the principal operation of the clustering algorithm. Therefore we use the terms objects and distances instead of transformations and rms deviations for the remainder of this section.

In pattern recognition, heuristic approaches to clustering are used frequently. These algorithms are quite fast, but have the disadvantages that they cannot be used with arbitrary distance functions and that the results are rarely well defined, as they depend on internal parameters or on the sequence in which the clustered objects are processed. In the present application, hierarchical clustering can be applied because the number of transformations is not that large. A survey of clustering algorithms can be found in Refs. 37–39. In our application, an object is a transformation with a set of three interactions. If two objects are clustered, their interactions should occur simultaneously. Thus, we want to avoid large distances between objects in the same cluster. We achieve this by using complete-linkage clustering, which means that the distance between any pairs of objects in a cluster is smaller than a given threshold τ.

In general, complete-linkage hierarchical clustering works as follows [37]. At the beginning, each object represents a singleton cluster. The distance between two clusters is defined as the maximum distance between the objects of the clusters. As long as the minimum distance between two clusters is less than a given threshold τ, the following procedure is repeated: determine the two clusters with minimum distance and merge them into a single cluster. Thus, the order in which clusters are merged is determined by the distance between the clusters.

TABLE 2
SET OF RECEPTORS USED IN TESTING THE FRAGMENT-PLACING ALGORITHM OF FLEXX

No.	Name	PDB code	Chain	1	2	3	4
1	Dihydrofolate reductase	4dfr	A	HOH 403, HOH 405	6.0	62	616
2	Beta trypsin	3ptb		–	5.0[a]	117	453
3	Lactate dehydrogenase	1ldm		NAD 1	6.5	75	395
4	*p*-Hydroxybenzoate hydrolase	2phh		–	7.5	138	409
5	Purine nucleoside phosphorylase	1ulb		–	8.0	168	787
6	Streptavidin	1stp		–	6.5	128	681
7	Ribonuclease A	6rsa[b]		–	6.5	106	473
8	Carboxypeptidase A	2ctc		ZN 308	6.5	113	585
9	Triosephosphate isomerase	5tim	A, B	–	8.0	122	738

Columns 1–4: (1) additional hetero-groups loaded; (2) active-site cutoff value; (3) number of heavy atoms of the active site; and (4) number of interaction points generated by FLEXX.
[a] Complete amino acids.
[b] Hydrogens removed. The protonated histidine template is assigned to His^{12} and His^{119}.

A naive algorithm would use a distance matrix with n^2 entries, where n is the number of objects. Because at most n merging operations are possible and the minimum of n^2 distances must be computed in each iteration, this algorithm has a run time proportional to n^3. Use of a binary heap to retrieve the minimum distance results in a run time of $O(n^2 \log n)$. This run time can be reduced further to $O(m \log n)$ by using a sparse representation of the distance matrix where m is the number of object pairs with distance smaller than τ, assuming that this set of pairs is already known. A detailed description of this algorithm can be found in Appendix 3.

Of course, we have to provide an algorithm that identifies all pairs of objects with a distance less than τ. The naive algorithm for this task runs in quadratic time. In our software, we use a heuristically enhanced version of this algorithm, which exhibits an acceptable performance. There are also more complicated methods for solving this problem in subquadratic time [40,41]. These techniques are not used here because of the larger implementation effort.

Computing clusters of transformations Let us return to the application of molecular docking. In this section, we discuss how to merge two possible placements of the ligand into a single placement. Aside from a transformation, we assign to each placement a list of matches. Each match represents an interaction between the receptor and the ligand, consisting of the interaction centers at the receptor and the ligand and the interaction point at the receptor (remember that the interaction surfaces at the receptor are approximated by discrete sets of interaction points, see Fig. 3).

A small rms deviation between the transformations of two placements means that the interactions of the two placements can take place simultaneously. Thus, the combination of the placements can be achieved by merging the lists of matches and recomputing a transformation by superposing the matched interaction centers of the ligand onto the assigned interaction points of the receptor. The superposition is done with the iterative algorithm of Ferro and Hermans [42]. Note that this recomputation of a superposition need not be done during the clustering algorithm, because the distances between clusters are based only on the distances between their objects. If two interactions share the same interaction surface but differ in their interaction points, the energetically less favorable interaction is deleted from the list of matches.

Results

We have tested the fragment-placing algorithm by reproducing some known receptor–ligand complexes from the Brookhaven Protein Data Bank [25]. In the case of larger ligands, a specific fragment of the ligand is selected interactively. Lists of receptors and ligands used in the test are given in Tables 2 and 3, respectively.

Preparing the input

The preparation of the receptor for FLEXX consists of two steps. First, the active site must be determined with a molecular modeling tool and stored in an active-site

TABLE 3
LIST OF LIGANDS USED IN THE TEST

No.	Name	1	2	3
1	Methotrexate	12	1	15
2	Benzamidine	9	2	11
3	Oxamate	6	2	6
4	*p*-Hydroxybenzoate	10	6	11
5	Guanine	11	1	20
6	Biotin	9	36	4
7	Uridine vanadate	11	1[a]	6
8	L-Phenyl lactate	5	15	4
9	Sulfate	5	1	4

Columns 1–3: (1) number of atoms; (2) number of discrete conformations; and (3) number of interaction geometries assigned. All values refer to the considered fragment only. The fragments are shown in Fig. 7.
[a] Held rigid because SCA cannot handle vanadium.

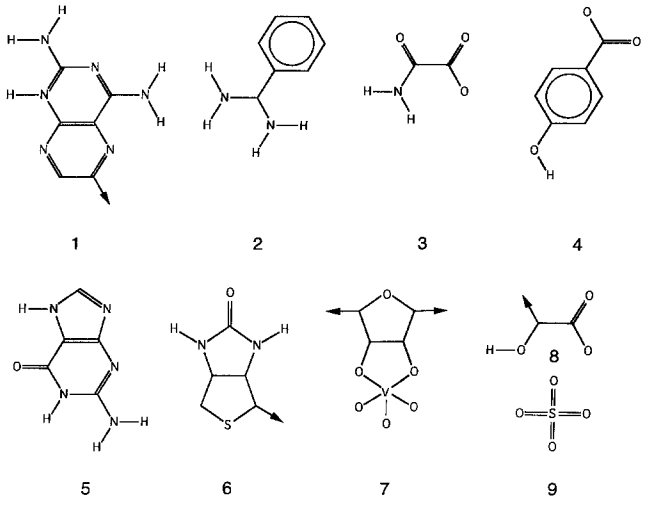

Fig. 4. Selected fragments of the ligands of the test systems. **1**: methotrexate; **2**: benzamidine; **3**: oxamate; **4**: *p*-hydroxybenzoate; **5**: guanine; **6**: biotin; **7**: uridine vanadate; **8**: L-phenyl lactate; and **9**: sulfate. The arrows mark the bonds where the rest of the molecule is attached to the base fragment.

file. We define the active site as comprising all atoms of the receptor whose distance to the crystallized ligand lies below a given cutoff. The cutoff varies in the examples because of the different shapes of the pockets. In the case of larger ligands, the complete ligand is used to define the active site, even if only one fragment of the ligand should be docked with FLEXX. The cutoff values are listed in Table 2.

Subsequently, the receptor description files are generated manually. In all test cases, the physicochemical properties of the receptor atoms are set to standard values. Water molecules are excluded, except for the conserved molecules HOH 403 and HOH 405 in dihydrofolate reductase. For adding polar hydrogens by FLEXX, the corresponding torsional angles are automatically set to standard values, except for the OH group in tyrosine. Here, we decide between the two possible in-plane positions by visual inspection of the complex. Hydrogens from included water molecules are placed with molecular modeling tools by visual inspection. Nonpolar hydrogens are not modeled explicitly (united atom radii model). The most important parameters and properties of the active site are listed in Table 2.

When the receptor is loaded into FLEXX for the first time, the surface atoms, as defined by Richards [43], are identified and stored in a surface file. This provision avoids expensive recomputation on the same receptor in subsequent runs. The process requires a run time of about 30 s on a Sun SPARCstation 20 workstation and is not contained in the following run time results.

The ligand is extracted from the PDB file. SYBYL [44] atom and bond types are assigned, hydrogens are added in standard geometries, and elementary charges are defined at the atoms. Finally, the ligand is minimized with the TRIPOS force field [44] using standard parameters and is stored in mol2 file format. Every time the ligand is loaded into FLEXX, the following operations are done automatically: identification of ring systems and rotatable bonds, computation of internal coordinates, computation of ring-system conformations with SCA [33], assignment of energetically favorable torsional angles to each rotatable bond (MIMUMBA database [26,32]), and assignment of interaction types and geometries. The resulting number of discrete conformations and the number of interaction groups identified for the ligands of the test set are listed in Table 3. As soon as the ligand is available in mol2 format, no further manual intervention is needed. This is an important feature if larger sets of ligands are to be docked automatically.

Parameters of the algorithm

In principle, the complete background knowledge of the program (see Methods section) can be viewed as a set of parameters. Thus, it is impossible to give a complete list of parameters here. We want to consider only the most important parameters, which are summarized in Table 4. Note that all tests have been performed with exactly the same set of parameters and static data files.

The most critical parameters for run time as well as for the quality of the results are the parameters controlling the interaction-point density and the matching accuracy.

TABLE 4
LIST OF MOST IMPORTANT PARAMETERS AND THEIR VALUES USED IN THE TEST

Parameter	Value
The interaction points are placed on the surface of the sphere, so that the arc length between two neighboring points is less than this value	1.2 Å
Tolerance for matching a triangle edge during the fragment-placing algorithm	0.4 Å
Minimum length of a triangle edge	1.5 Å
Maximum length of a triangle edge	10.0 Å
Maximum allowed distance tolerance between matched interactions	0.55 Å
Maximum allowed angle tolerance between matched interactions	0.1 rad
Maximum rms distance between two solutions assigned to the same cluster in the fragment-placing algorithm	1.7 Å
Maximum allowed overlap volume between a receptor atom and an atom of the placed fragment	2.5 $Å^3$
Maximum allowed overlap volume averaged over the fragment atoms	1.0 $Å^3$

TABLE 5
RESULTS OF THE NINE DOCKING EXPERIMENTS

No.	No. of solutions	Run time		1		2			3			4
		Preprocessing	Docking	a	b	a	b	c	a	b	c	
1	18	14.54	26.52	−26.7	0.73	−26.7	0.73	1	−24.3	0.48	4	−55.3[a]
2	31	14.18	13.48	−26.5	0.53	−26.5	0.53	1	−26.5	0.53	1	−27.2
3	10	12.08	15.20	−35.3	1.53	−33.6	0.66	2	−33.6	0.66	2	−30.8
4	61	12.93	67.48	−24.6	1.82[b]	−23.7	0.52	5	−23.5	0.22	6	−26.7
5	54	35.03	72.59	−19.3	0.66	−19.3	0.66	1	−15.2	0.59	5	−30.2
6	30	20.68	18.06	−22.2	2.67[c]	−21.9	1.07	3	−12.0	0.75	25	−76.4[a]
7	48	7.36	91.54	−31.3	5.56	−25.7	0.46	6	−25.7	0.46	6	−28.5[a]
8	29	24.65	86.23	−27.7	1.40	−27.2	0.57	3	−27.2	0.57	3	−22.2[a]
9	43	22.25	3.11	−16.5	1.23[d]	−14.8	1.05	6	−6.5	0.50	42	−13.1

Columns 1–4: (1) highest ranking solution; (2) solution of highest rank with an rms deviation ≤1.1 Å; (3) solution with minimal rms deviation; and (4) experimentally observed free energy of binding. Subcolumns: (a) estimation of free energy of binding; (b) rms deviation; and (c) rank of the solution. The times are given in s.ms; the test was performed on a Sun SPARCstation 20. The free energies are given in kJ/mol and the rms deviations in Å.

[a] Experimental energy of the whole ligand, not of the base fragment alone.

[b] Symmetric solution.

[c] If the ring conformation found in the crystal is used, a solution with an rms deviation of 0.49 Å and an energy estimation of −24.043 kJ/mol is found as the highest ranking.

[d] Only the sulfur atom is considered because of the symmetry of the oxygens.

Both parameters depend on each other in an obvious way. Clearly, a low point density leads to short run times, but at the same time the possibility of losing the correct matching increases because the interactions often come to have distorted geometries. This problem can only be avoided by applying energy-minimization techniques, which, in turn, are expensive in run time.

The output

The output of FLEXX is a set of predicted binding modes, with for each an estimated binding energy. Each binding mode consists of the conformation of the ligand and its orientation relative to the receptor.

The results we obtained from FLEXX for docking the fragments of Fig. 4 are listed in Table 5. Column 2 contains the number of different placements produced by FLEXX. Usually, this number is on the order of a few dozens if the ligand fits the receptor site. Note that no energy cutoff is used to reduce the number of placements. Despite the clustering at the end of the fragment-placing algorithm, some of the placements are quite similar. The reason for this phenomenon is that, after the clustering step, the ligand is resuperposed on the interaction points. A second clustering of the solutions can be used to eliminate similar solutions. Thus, normally only a few possible binding modes per conformation are finally returned.

Columns 3 and 4 in Table 5 give the run time of FLEXX including the placement of all conformations for each docking experiment. The test has been performed on a Sun SPARCstation 20. The preprocessing time contains the complete preparation of the data for the receptor and the ligand, as described above. The most time-consuming step is the initialization of the hash tables, which requires about 90% of the preprocessing time. The run time of the computation of ring-system conformations by SCA, which is normally about 10–20 s depending on the complexity of the ring systems, has not been taken into account here.

The docking time (column 4 in Table 5) is the run time for the fragment-placing algorithm. Averaged over the nine docking experiments, we obtain a run time of about 6 s per conformation. It can be expected that the run time is smaller for fragments with an inferior steric or physicochemical fit.

Columns 1a and 1b in Table 5 give the free energy estimation computed with our version of Böhm's energy function [31] and the rms deviation of the highest ranking placement produced by FLEXX. Only in the cases of biotin (**6**) and uridine vanadate (**7**) the predicted binding

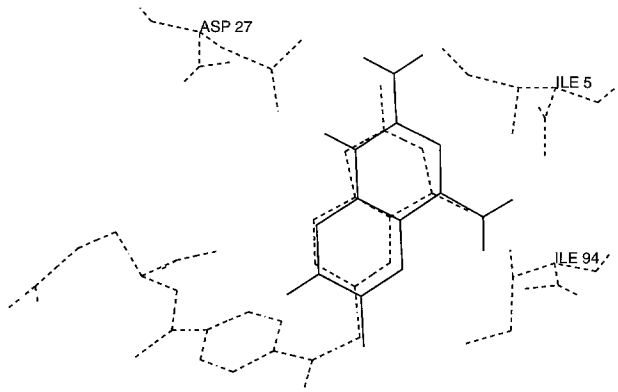

Fig. 5. Solution number 4 of the test system dihydrofolate reductase–methotrexate (solution with lowest rms deviation). The crystal is shown in dashed lines; the predicted placement in solid lines (picture created with WHATIF [45]).

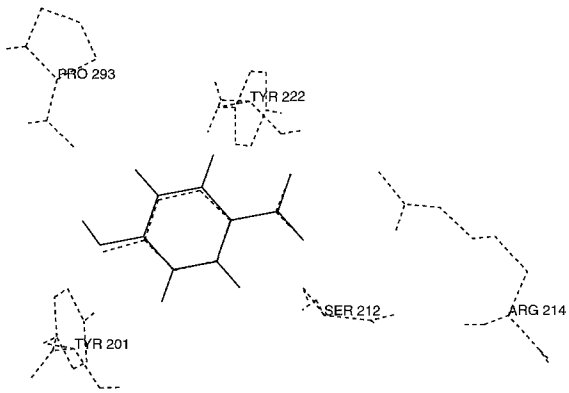

Fig. 6. Solution number 1 of the test system *p*-hydroxybenzoate hydrolase–*p*-hydroxybenzoate (solution with lowest estimated energy). The crystal is depicted in dashed lines; the predicted placement in solid lines (picture created with WHATIF [45]).

mode is substantially different from the crystal. This is not surprising, because the placement as proposed by the crystal structure is the energetically most favorable placement for the whole ligand, not just for the fragment considered by the fragment-placing algorithm.

In Table 5, columns 2a–3c analyze placements with a low rms value. Columns 2a–c contain the estimated free energy, the rms deviation, and the rank (sorted by the estimated energy) of the energetically highest ranking solution, which is highly similar (rms deviation ≤ 1.1 Å) to the crystal structure. Columns 3a–c contain the estimated free energy, the rms deviation, and the rank of the solution with the lowest rms with respect to the crystal structure. In all cases, a placement with low rms deviation is found among the six highest ranking placements. Three placements with low rms deviations are shown in Figs. 5–7. The last column (4) shows the experimentally observed binding energy. Even though Böhm's energy function [31] only roughly approximates the free binding energy, we find that if we dock complete ligands, our free energy estimates end up sufficiently close to the observed energy to be of service in the docking procedure. Note that in examples 1 and 6–8, the value in column 4 is the observed binding energy for the whole ligand and not that for the docked fragment.

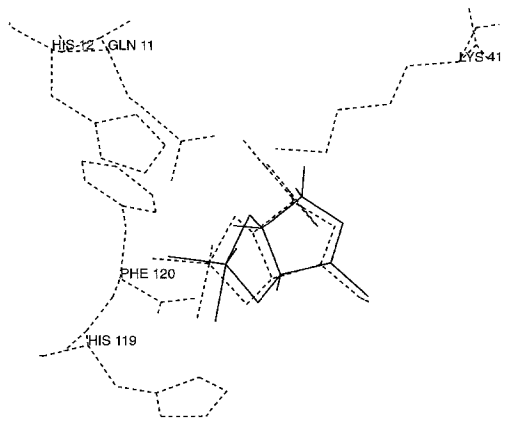

Fig. 7. Solution number 6 of the test system ribonuclease A–uridine vanadate (solution with lowest rms deviation). The crystal is depicted in dashed lines; the predicted placement in solid lines (picture created with WHATIF [45]).

Conclusions

We have presented a new algorithm for placing molecular fragments into active sites of proteins. The algorithm is based on a strategy similar to the pose-clustering technique used in the area of pattern recognition. For each triangle of interaction centers of the ligand, approximately congruent triangles are constructed from the set of interaction points in the active site of the receptor. From each such match, a superposition with minimal rms deviation is computed. These transformations are clustered to fragment placements with a complete-linkage hierarchical clustering algorithm.

We have shown that our algorithm is able to reconstruct complexes known from X-ray crystallography, assuming that the receptor is given in the bound conformation. In all cases, a solution with an rms deviation below 1.1 Å is found among the six highest ranking solutions. Thus, the algorithm can be used for docking small ligands or ligand fragments as part of a de novo design tool or a docking tool like FLEXX, which is under development at GMD.

The most serious restriction in currently available docking and de novo design algorithms, as well as in our approach, is the rigidity of the receptor. However, there are two reasons why we think that the fragment-placing algorithm is a promising starting point to overcome this limitation. First, there are no elements requiring the rigidity of the receptor, like in grid-based energy calculations, which are very popular in time-efficient docking algorithms [4,5,20]. Second, the algorithm is fast enough to handle a larger combinatorial complexity.

Acknowledgements

We thank Bernd Kramer for supporting the chemical modeling in FLEXX and Christoph Bernd and Claus Hiller, who have done parts of the implementation. Likewise, it is our pleasure to thank our partners in the RELIWE Project, especially Gerhard Klebe, Thomas Mietzner, Joachim Böhm and Hugo Kubinyi from BASF main laboratory for preparing parts of the input data, making MIMUMBA available to us, and introducing us to the pitfalls of molecular modeling in many stimulating discussions. We thank Gert Vriend for making the WHATIF modeling package [45] available to us. The Reliwe Project is partially funded by the German Federal Ministry for Education, Science, Research and Technology (BMBF) under Grant 01 IB 302 A.

References

1 Kuntz, I.D., Blaney, J.M., Oatley, S.J., Langridge, R.L. and Ferrin, T.E., J. Mol. Biol., 161 (1982) 269.
2 Kuhl, F.S., Crippen, G.M. and Friesen, D.K., J. Comput. Chem., 5 (1984) 24.
3 DesJarlais, R.L., Sheridan, R.P., Seibel, G.L., Dixon, J.S., Kuntz, I.D. and Venkataraghavan, R., J. Med. Chem., 31 (1988) 722.
4 Meng, E.C., Shoichet, B.K. and Kuntz, I.D., J. Comput. Chem., 13 (1992) 505.
5 Lawrence, M.C. and Davis, P.C., Protein Struct. Funct. Genet., 12 (1992) 31.
6 Shoichet, B.K. and Kuntz, I.D., Protein Eng., 6 (1993) 723.
7 DesJarlais, R.L., Sheridan, R.P., Dixon, J.S., Kuntz, I.D. and Venkataraghavan, R., J. Med. Chem., 29 (1986) 2149.
8 Eisen, M.B., Wiley, D.C., Karplus, M. and Hubbard, R.E., Protein Struct. Funct. Genet., 19 (1994) 199.
9 Gubernator, K., lecture presented at the conference Bioinformatik – Computereinsatz in den Biowissenschaften, Jena, 1994.
10 Sandak, B., Nussinov, R. and Wolfson, H.J., Comput. Appl. Biosci., 11 (1995) 87.
11 Böhm, H.-J., J. Comput.-Aided Mol. Design, 6 (1992) 61.
12 Moon, J.B. and Howe, W.J., Protein Struct. Funct. Genet., 11 (1991) 314.
13 Böhm, H.-J., J. Comput.-Aided Mol. Design, 6 (1992) 593.
14 Gillet, V.J., Johnson, P. and Sike, S., Tetrahedron, 3(6C) (1990) 681.
15 Gillet, V.J., Newell, W., Mata, P., Myatt, G., Sike, S., Zsoldos, Z. and Johnson, A.P., J. Chem. Inf. Comput. Chem., 34 (1994) 207.
16 Rotstein, S.H. and Murcko, M.A., J. Comput.-Aided Mol. Design, 7 (1993) 23.
17 Leach, A.R. and Kuntz, I.D., J. Comput. Chem., 13 (1992) 730.
18 Lewis, R.A. and Leach, A., J. Comput.-Aided Mol. Design, 8 (1994) 467.
19 Goodsell, D.S. and Olson, A.J., Protein Struct. Funct. Genet., 8 (1990) 195.
20 Mizutani, M.Y., Tomioka, N. and Itai, A., J. Mol. Biol., 243 (1994) 310.
21 Fischer, D., Lin, S.L., Wolfson, H.L. and Nussinov, R., J. Mol. Biol., 248 (1995) 459.
22 Rarey, M., Kramer, B. and Lengauer, T., In Rawlings, C. (Ed.) Proceedings of the Third International Conference on Intelligent Systems in Molecular Biology, AAAI Press, Menlo Park, CA, 1995, pp. 300–308.
23 Delaney, J.S., J. Mol. Graphics, 10 (1992) 174.
24 Del Carpio, C.A., Takahashi, Y. and Sasaki, S., J. Mol. Graphics, 11 (1993) 23.
25 Bernstein, F.C., Koetzle, T.F., Williams, G.J.B., Meyer Jr., E.F., Brice, M.D., Rodgers, J.R., Kennard, O., Shimanouchi, T. and Tasumi, M., J. Mol. Biol., 112 (1977) 535.
26 Klebe, G. and Mietzner, T., J. Comput.-Aided Mol. Design, 8 (1994) 583.
27 Klebe, G., J. Mol. Biol., 237 (1994) 221.
28 Taylor, R. and Kennard, O., Acc. Chem. Res., 17 (1984) 320.
29 Murray-Rust, P. and Glusker, J.P., J. Am. Chem. Soc., 106 (1984) 1018.
30 Tintelnot, M. and Andrews, P., J. Comput.-Aided Mol. Design, 3 (1989) 67.
31 Böhm, H.-J., J. Comput.-Aided Mol. Design, 8 (1994) 243.
32 Klebe, G. and Mietzner, T., In Jones, D.W. (Ed.) Organic Crystal Chemistry, Oxford University Press, Oxford, 1992, pp. 135–158.
33 Hoflack, J. and De Clercq, P.J., Tetrahedron, 44 (1988) 6667.
34 Linnainmaa, S., Harwood, D. and Davis, L.S., IEEE Trans. Pattern Anal. Machine Intell., 10 (1988).
35 Olson, C.F., In IEEE Conference on Computer Vision and Pattern Recognition, 1994, pp. 251–258.
36 Lamdan, Y. and Wolfson, H.J., Proceedings of the IEEE International Conference on Computer Vision, 1988, pp. 238–249.
37 Duda, R.O. and Hart, P.E., Pattern Classification and Scene Analysis, Wiley, New York, NY, 1973.
38 Murtagh, F., Comput. J., 26 (1983) 354.
39 Dorndorf, U. and Pesch, E., ORSA J. Comput., 6 (1994) 141.
40 Preparata, F.P. and Shamos, M.I., Computational Geometry, Springer, New York, NY, 1985.
41 Lenhof, H.-P., Ph.D. Thesis, Universität Saarbrücken, Saarbrücken, 1993.
42 Ferro, D.R. and Hermans, J., Acta Crystallogr., A33 (1977) 345.
43 Richards, F.M., Annu. Rev. Biophys. Bioeng., 6 (1977) 151.
44 SYBYL, Tripos Associates, Inc., St. Louis, MO, 1994.
45 Vriend, G., J. Mol. Graph., 8 (1990) 52.

Appendices

1. Searching for nearly congruent triangles

In this appendix, we give a more formal description of our data structure for the problem of finding similar triangles. First, we have to define what we mean by calling two triangles similar.

Definition 1 A triangle (p_0,p_1,p_2) with points p_0,p_1,p_2 having types $c(p_i)$ from the discrete set C, $i \in \{0,1,2\}$ is called a typed triangle. Typed line segments are defined analogously.

Definition 2 Two triangles, each defined by a triple of points in $\mathbb{R}^3$, (p_0,p_1,p_2) and (q_0,q_1,q_2), are δ-congruent if and only if for each $i \in \{0,1,2\}$: $\|p_i - p_{i+1 \bmod 3}| - |q_i - q_{i+1 \bmod 3}\| \leq \delta$. For line segments, δ-congruence is defined analogously. Two typed triangles (line segments) are δ-compatible if and only if they are δ-congruent and $c(p_i) = c(q_i)$ for $i = 0,1,2$.

The problem can now be stated as follows: given a discrete set of points $P \in \mathbb{R}^3$ with types $c(p) \in C$ for each $p \in P$ and a query triangle (q_0,q_1,q_2) with types $c(q_i) \in C$, find all δ-compatible triangles (p_0,p_1,p_2) that can be composed of points in the set P.

The data structure consists of a hash table for each combination of two types. All line segments that can be constructed from P are stored in the hash table designa-

```
TRIANGLE_QUERY (triangle (q0,q1,q2), type function c)
% query algorithm for finding all δ-compatible triangles out of a set of points P,
% based on line-segment hashing
1  L1 ← line_segment_query ((q0,q1),c); L2 ← line_segment_query ((q0,q2),c);
2  cur2 ← head(L2);
3  while (L1 ≠ ∅ ∧ L2 ≠ ∅) do
4     (pi,pj) ← ↑head(L1); (p'i, pk) ← ↑cur2;
5     case
6        i < i':  remove_head(L1); cur2 ← head(L2);
7        i > i':  remove_head(L2); cur2 ← head(L2);
8        i = i':  if ||q1 − q2| − |pj − pk|| ≤ δ then
9                    output triangle (pi,pj,pk); fi;
% go to the next pair of list elements
10             cur2 ← next(cur2);
11             if cur2 = end_of_list(L2) then
12                remove_head(L1); cur2 ← head(L2); fi;
13 esac; od;
```

Fig. 8. Triangle query algorithm based on line-segment hashing. The function line_segment_query() represents the line-segment hashing and produces a list of δ-compatible line segments out of the stored set of points.

ted by the types of its end points. The length of a line segment $d = |p_0 - p_1|$ defines the bucket $l \in \mathbb{N}_0$ by the inequalities $2\delta l \le d \le 2\delta(l+1)$ and $h(l)$ is the hash address of the line segment, where h is an adequate hash function. In practical parameterizations, the number of buckets is sufficiently small to use the length of the line segment directly for addressing the buckets, i.e., the identity function is used for h. All line segments assigned to the bucket l are stored in a list $L(l)$.

The retrieval of all line segments matching a given edge of a triangle with length d can be done easily by scanning the buckets l and l', where l is defined as above and $l' = l+1$ if $d > 2\delta(l+1/2)$, $l' = l-1$ otherwise. With this data structure in mind, we can now develop an efficient query algorithm for triangles.

Assume that we are given a query triangle (q_0,q_1,q_2) with types $c(q_i)$, $i \in \{0,1,2\}$ for the points q_i. As a first step, we retrieve from the hash tables two lists of matching line segments for two sides of the query triangle. Denote the list for the line segment (q_0,q_1) with $L(q_0,q_1)$ and the list for the line segment (q_0,q_2) with $L(q_0,q_2)$. The remaining task is to construct all δ-congruent triangles out of the line segments in these two lists.

Assume that the lists $L(q_0,q_1)$ and $L(q_0,q_2)$ are sorted by increasing index i of the point p_i matching q_0. How this can be achieved efficiently will be explained later. The set of all δ-congruent triangles can now be generated by a kind of list-merging operation based on the following obvious lemma:

Lemma 1 Let P be a point set with a type c(p) for each $p \in P$ and let (q_0,q_1,q_2) be a query triangle with types $c(q_i)$. $(p_i,p_j,p_k) \in P^3$ is δ-compatible with (q_0,q_1,q_2) if and only if (p_i,p_j) is δ-compatible with (q_0,q_1), (p_i,p_k) is δ-compatible with (q_0,q_2) and the line segments (q_1,q_2) and (p_j,p_k) are δ-congruent.

The merging operation runs through the two lists $L(q_0, q_1)$ and $L(q_0,q_2)$ successively. Let (p_i,p_j) and (p'_i,p_k) be the current list elements in $L(q_0,q_1)$ and $L(q_0,q_2)$, respectively. If $p_i = p'_i$, then (p_i,p_j,p_k) is a possible candidate for a δ-compatible triangle. In order to verify that the candidate triangle is indeed δ-compatible with (q_0,q_1,q_2), the remaining distance condition $||q_1 - q_2| - |p_j - p_k|| \le \delta$ must be checked.

The merging operation is summarized in Fig. 8. The

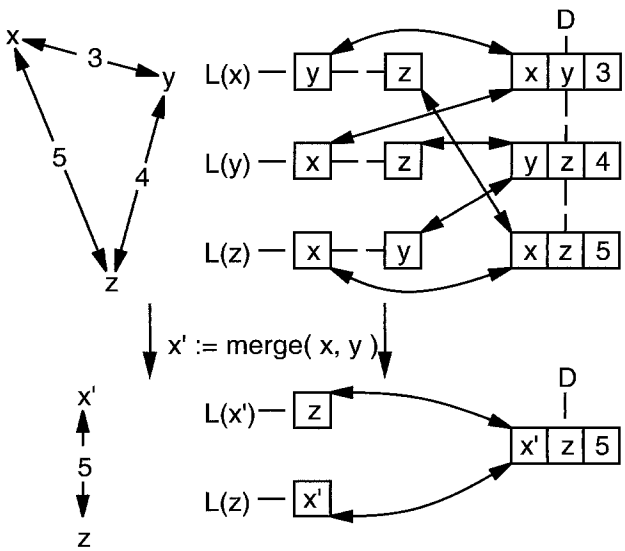

Fig. 9. One step in the clustering algorithm. The example contains the three clusters x,y,z. On the right-hand side the cluster candidate list D and the adjacency list L of each cluster are shown. The situation after clustering x and y is shown in the lower part.

correctness of the algorithm is a direct implication of lemma 1. The run time increases linearly with the sum of the length of the two lists processed.

As mentioned above, the merging operation requires sorted input lists. The sorting can be done in the preprocessing phase on the list in every bucket. Remember that the input list $L(q_0,q_1)$ is combined out of the two buckets l, l'. Thus, producing a sorted input list starting with the sorted lists of the buckets $L(l)$ and $L(l')$ can be achieved by list merging in linear time.

2. Distance between transformations

The distance measure used in FLEXX for clustering the transformations is the rms deviation between different placements of the fragment. Let $T_i=(R_i,t_i)$, $i=1,2$ be two transformations of the fragment, consisting of n atoms with coordinates x_i, $i\in\{1,...,n\}$. Then the distance is defined as

$$d_{rms}(T_1,T_2)=\sqrt{\frac{1}{n}\sum_{i=1}^{n}((R_1x_i+t_1)-(R_2x_i+t_2))^2}$$

For a fixed set of vectors $x_1,...,x_n$, where $x_k(i)$ denotes the ith component of vector x_k (in our application, only the transformations are variable), the rms deviation can be computed in constant time using the formula (see also Ref. 42):

In Eq. A2-1:

$$A=(a_{ij})=R_1^tR_2$$

$$X_m=\frac{1}{n}\sum_{k=1}^{n}x_k$$

$$X_{ij}=\frac{1}{n}\sum_{k=1}^{n}x_k(i)x_k(j)$$

and

$$\delta_{ij}=\begin{cases}1 & \text{if } i=j\\ 0 & \text{if } i\neq j\end{cases}$$

$$d_{rms}(T_1,T_2)=\sqrt{2\sum_{i=1}^{3}\sum_{j=1}^{3}(\delta_{ij}-a_{ij}-a_{ji})X_{ij}+(t_1-t_2)^2+2(t_1-t_2)^t(R_1-R_2)X_m} \qquad \text{(A2-1)}$$

3. Time-efficient complete-linkage clustering

Here, we present a complete-linkage hierarchical clustering algorithm with run time $O(m \log n)$ using a sparse representation of the distance matrix, where m is the number of object pairs with distance smaller than the distance threshold τ, assuming that this set of pairs is already known.

The clustering algorithm is based on the following data structure. A cluster is represented by its set of objects. Note that the choice of a representative object inside a cluster is not necessary, because the distance between clusters is based on distances between the objects of the clusters only. All pairs of clusters with a distance smaller than τ are stored in a doubly linked candidate list D, which is sorted by increasing distance. These pairs of clusters are called the cluster candidates. For each cluster C, a list L(C) of all clusters that are at most τ apart from C is generated. L(C), called the adjacency list, is sorted increasingly by a unique cluster id number. Thus, for each element C' in L(C), we have one element (C,C') in D that is cross-linked to the elements in L(C) and L(C') (see Fig. 9). If the m pairs of clusters with distance smaller than τ are known, this list can be constructed in $O(m \log n)$ time with a standard sorting algorithm.

The clustering algorithm proceeds as follows: while the candidate list D is not empty, take the head (C_1,C_2) of D, merge the clusters C_1 and C_2 and update the data structure D,L(C) for $C=C_1,C_2,C_1\cup C_2$. Maintaining the data structures can be achieved by a merging operation on the lists of the combined clusters $L(C_1)$ and $L(C_2)$.

Let $C_{new}=C_1\cup C_2$ be the new cluster. The update of the data structure entails the deletion of cluster candidates containing one of the old clusters C_1,C_2 from D, the generation of cluster candidates containing C_{new}, the generation of the list $L(C_{new})$ and the deletion of the lists $L(C_1)$ and $L(C_2)$. Because of the cross-linking between the lists $L(C_1)$, $L(C_2)$ and D, the deletion steps can be done in linear time in the length of lists $L(C_1)$ and $L(C_2)$. Note that, for a cluster candidate (C_1,C') in D, the element containing C_1 in the list of L(C') can be found and deleted in constant time with the aid of the cross-link.

Before removing the lists $L(C_1)$ and $L(C_2)$, we use them to generate the cluster candidates containing the new cluster C_{new}. Let (C_{new},C') be a new cluster candidate, i.e., $d(C_{new},C')\leq\tau$. Because C_{new} is the union of C_1 and C_2, $d(C_{new},C')=\max\{d(c,c') \mid c\in C_{new},c'\in C'\}=\max\{d(C_1,C'),d(C_2,C')\}$ and thus $d(C_1,C')\leq\tau$ and $d(C_2,C')\leq\tau$ holds. This implies that (C_1,C') and (C_2,C') were cluster candidates in the last iteration. Thus, the new cluster candidates are

```
COMPLETE_LINKAGE_CLUSTERING (object set C, distance function d, distance threshold τ)
% computes a complete linkage clustering of the objects in C
1  initialize sorted list D of cluster candidates and the adjacency lists L(C); each C ∈ C;
2  new ← |C| + 1;
3  while D ≠ ∅ do
4    (C1,C2) ← ↑head(D); Cnew ← C1 ∪ C2;
% update the lists D, L(C)
5    while L(C1) ≠ ∅ ∧ L(C2) ≠ ∅ do
6      Ci ← ↑head(L(C1)); Ci' ← ↑head(L(C2));
7      case
8        i < i':  remove Ci from L(C1); remove C1 from L(Ci);
9                 remove (Ci,C1) from D;
10       i > i':  remove Ci' from L(C2); remove C2 from L(Ci');
11                remove (Ci',C2) from D;
12       i = i':  insert (Cnew,Ci, max{d(C1,Ci), d(C2,Ci)}) in D;
% (Ci,Cnew) is a new cluster candidate
13           insert Cnew in L(Ci); insert Ci in L(Cnew);
14           remove Ci from L(C1) and L(C2);
15           remove C1 and C2 from L(Ci);
16           remove (Ci,C1) and (Ci,C2) from D;
17   esac; od;
18   new ← new + 1; od;
```

Fig. 10. Complete linkage clustering algorithm, based on list merging techniques.

exactly those pairs (C_{new},C') with $C' \in L(C_1) \cap L(C_2)$. The remaining task is to maintain the ordering of the lists D and L(C) for all clusters C. The list $L(C_{new})$ is already sorted because of the correct ordering in the merged lists $L(C_1)$,$L(C_2)$. The new cluster gets the next higher index, i.e., max{index of a cluster} + 1. Thus, a new entry C_{new} in a list L(C') can be appended to the end of L(C'). Let (C_{new},C') be a new cluster candidate and let $d(C_1,C') \geq d(C_2,C')$ without loss of generality. Then, $d(C_{new},C') = d(C_1,C')$ and the ordering of D can be maintained by inserting the new cluster candidate in the place of the old candidate (C_1,C') in list D. The complete algorithm is shown in Fig. 10.

Once the sorted list of cluster candidates is initialized, the algorithm has an asymptotic run time of O(m). Thus, the total run time of the clustering step is $O(m \log n)$.