Structural Alignment

Introduction

Problem Definition: Given two configurations of points in the three dimensional space, find those rotations and translations of one of the point sets which produce "large" superimpositions of corresponding 3-D points. Notice that we are not necessarily looking for the largest match atom-wise, but perhaps the most meaningful one. For example – matching domains or functional regions. As we have seen before, least RMSD calculation requires a correspondence between pairs of atoms. This is a major task in structural comparison – the correspondence between two matching proteins. When the two conformations come from the same protein – we simply use the lRMSD formula and measure distance. Otherwise – this is a difficult problem. So the question asked – how do you compare two different proteins? In other words – find the optimal (sub)structural alignment of two proteins.

Global vs. Local Alignment While global structural alignment aims to find the best overall alignment (good for protein classification), local alignment aims to find the best correspondence between a *motif*, a small substructure of a protein, often between 3 and 20 amino acids assumed to have a biological or functional significance, and a target, which is a full protein structure.

Sequence Order Dependence: Sequence order dependent alignment aligns two sequences so that the match follows the amino acid order. This is a 3-D curve matching – an inherently 1-D task. Sequence order independent alignment is a "real" 3-D task. The structures are considered a set of 3D points in space, and are matched without considering the amino acid order. This enables the detection of non-sequential motifs in proteins, e.g. molecular surface motifs and especially, similar binding sites. Sequence order independent alignment allows us to search of structural databases with only partial and disconnected structural information which can have significant applications in drug design. Motifs preserving sequence order might be biologically more meaningful than similar size non-sequential motifs. The computational task becomes much more complex when sequence order is not exploited.

Structure Alignment Algorithms:

DALI: Distance matrix ALIgnment

The algorithm [**?**] is based on the fact that similar 3D structures have similar intra-molecular distances. Given two proteins A and B, a distance matrix is created for each one of the compared proteins with all pairwise distances, such that $D[i.j] = dist(i, j)$. The distances are between the $C\alpha$ atoms. The two distance matrices are then aligned. The optimal alignment is determined by a score as follows:

$$S = \sum_{i=1}^{L} \sum_{j=1}^{L} \phi^R(i, j)$$

Where

$$\phi^R(i,j) = \theta^R - |D^A_{ij} - D^B_{ij}|$$

θ is a constant, $|D^A_{ij} - D^B_{ij}|$ the absolute differences between the two contact matrices after alignment. The implementation breaks up each matrix into sub-matrices of fixed size (6 by default), pair-up similar sub-matrices (one from each protein) and store the matching pairs in a list. Then assemble the sub-matrix pairs to get the overall alignment. Assembly is not a trivial problem. In earlier versions of the algorithm used Monte Carlo optimization. Later versions used branch-and-bound. The advantage of intra-molecular distances is that it does not require the two molecules to be aligned to begin with.

DISCO

This is an approach towards finding the Largest Common Point set [**?**]. DISCO operates on two ligand structures (small molecules) by generating a graph along the following definition:

- Graph Nodes: For a node a, all pairs of points a_1, a_2, with a1 from ligand 1, a_2 from ligand 2.
- Graph Edges: An edge (a,b) exists if the pairs (a_1, a_2) and (b_1, b_2) can be aligned simultaneously. (i.e. the distance between a_1 and b_1, and a_2 and b_2 is very similar)

This means that for a DISCO graph G, finding a clique implies finding a set of reasonably congruent points common to both ligand structures. Finding the largest clique is a well known NP complete problem. This is a significant problem, but because ligand structures often have very few points, computation of the largest clique in G is often tractable. To this end, a standard clique detection algorithm due to Bron and Kerbosch branch-and-bound technique can be applied to detect cliques in G. In addition, if multiple ligands are available then one can be chosen as a reference, and the rest compared to it.

TM-Align

This method [**?**] tries to align the $C\alpha$ atoms of the two input proteins as follows: There are three kinds of initial alignments: The first type is obtained by aligning the secondary structures of two proteins using dynamic programming. The element of the score matrix is assigned to be 1 or 0 depending on whether or not the secondary structure elements of aligned residues are identical. A gap opening penalty of 1 is assigned. For a given residue, a secondary structure is assigned to one of three states (α, β or coil) based on the $C\alpha$ coordinates of five neighboring residues. The i^{th} residue is assigned to $\alpha(\beta)$ if:

$$|d_{j,j+k} - \lambda_k^{\alpha(\beta)}| < \delta^{\alpha(\beta)}, (j = i-2, i-1, i; k = 2, 3, 4)$$

is satisfied for all $d_{j,j+k}$, the $C\alpha$ distance between the j^{th} and $(j+k)^{th}$ residues; otherwise, it is assigned to be a coil. There are eight parameters overall – λ_k^α and λ_k^β for $k = 2, 3, 4$, δ^α and δ^β. The parameters are optimized based on 100 non-homologous training proteins by maximizing the secondary structure assignment similarity to the DSSP definition, which defines the secondary structure elements on the basis of hydrogen bond patterns and requires the full set of backbone atomic coordinates. The second type of initial alignment is based on the gapless matching of two

structures. A gapless threading of the smaller structure is done against the larger structure, and the alignment with the best TM-score is selected. As a reminder, the TM-score is defined as:

$$TM_{score} = \max\{\frac{1}{L_{target}} \sum_{i=1}^{L_{aligned}} \frac{1}{1 + (\frac{D_i}{D_0(L_{target})})}\}$$

Where L_{target} and $L_{aligned}$ are the lengths of the target protein and the aligned region respectively. D_i is the distance between the i^{th} pair of residues and $D_0(L_{target}) = 1.24\sqrt[3]{L_{target} - 15} - 1.8$ is a normalization factor derived from an analysis of a large scale of structures. The third initial alignment is also obtained by DP using a gap-opening penalty of 1, but the score matrix is a half/half combination of the secondary structure score matrix and the distance score matrix selected in the second initial alignment.

The initial alignments are then submitted to a heuristic iterative algorithm: The structures are first rotated by the TMscore rotation matrix based on the aligned residues in the initial alignments. The score similarity matrix is defined as

$$S(i,j) = \frac{1}{1 + d_{ij}^2/d_0(L_{min}^2)}$$

where d_{ij} is the distance of the i^{th} residue in structure A and the j^{th} residue in structure 2 under the TM-score superposition and $D_0(L_{min}) = 1.24\sqrt[3]{L_{target} - 15} - 1.8$. A new alignment can be obtained by implementing DP on the matrix S(i, j) with an optimal gap opening penalty of -0.6. The structures are then superimposed again by the TM-score rotation matrix according to the new alignment and obtain a newer alignment by implementing DP with the new score matrix. The procedure is repeated until the alignment becomes stable and the alignment with the highest TM-score is returned. The alignments usually converge very fast, typically after 2-3 iterations.

SSAP (Sequential structure alignment program

The sequential structure alignment program (SSAP) method uses double dynamic programming to produce a structural alignment based on atom-to-atom vectors in structure space. Instead of the $C\alpha$ the vectors are built from $C\beta$ for all residues except glycine (which has no $C\beta$). This way the method which takes into account the rotameric state of each residue as well as its location along the backbone. SSAP works by first constructing a series of inter-residue distance vectors between each residue and its nearest non-contiguous neighbors for each protein. A series of matrices are then constructed containing the vector differences between neighbors for each pair of residues for which vectors were constructed. Dynamic programming applied to each resulting matrix determines a series of optimal local alignments which are then summed into a "summary" matrix to which dynamic programming is applied again to determine the overall structural alignment.

SSAP originally produced only pairwise alignments but has since been extended to multiple alignments as well. It has been applied in an all-to-all fashion to produce the hierarchical fold classification scheme for the CATH database.

Generally, SSAP scores above 80 are associated with highly similar structures. Scores between 70 and 80 indicate a similar fold with minor variations. Structures yielding a score between 60 and 70 do not generally contain the same fold, but usually belong to the same protein class with common structural motifs.

MatchMaker

The MatchMaker extension of Chimera constructs pairwise sequence alignments and uses them to superimpose the structures. The fit can be improved iteratively by pruning residue pairs far apart in space Given a superimposed set of two or more protein structures, Match → Align constructs a corresponding sequence alignment. Residue types are not used, only the spatial proximities of C-α. The user specifies a cutoff distance and a column inclusion criterion. In the pairwise case, a dynamic programming algorithm is used to determine the sequence alignment that best represents the structural alignment. Otherwise, heuristics are used.

Geometric Hashing

This is an object recognition/pattern matching method originally taken from computer vision [**?**]. The motivation for the original method is trying to recognize an object in a scene or an image. The method has two stages:

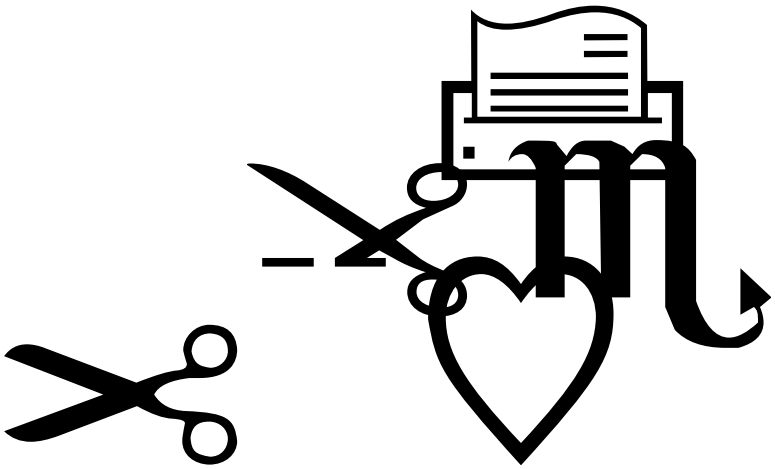
1. Preprocessing stage: learn a model pattern.
2. Recognition stage: the system is exposed to a new pattern of points, the Target, from which it is to identify a subset of reasonable geometric similarity to the model.

Here is an example. Given an image with multiple objects (Fig. 1 a.) We want to identify the pair of scissors in the image. We have a description of the scissors in some database (Fig. 1 b.). However, things may not be as simple. The scissors in the image may be rotated, scaled and even partially occluded. In some cases the scissors may be open or closed, but we will deal with that later. For now, let us assume that the scissors in the image may differ from the scissors in our database only by scale, rigid transformation (rotation and/or translation), and it may be partially occluded by other objects. Mathematically speaking, our goal is to find parts of the image that contain possible *transformed* copies of the scissors in the database.

To illustrate the process, we'll continue with a simpler example. Through this example we will show how the objects, both model and target, are represented and manipulated. We will start with a 2D example. Let us say our model is a set of points as seen in Fig. 2 a, and a query image (Fig. 2 b) and we want to search for a (possibly transformed) copy of the set of points in the query that maximizes the overlap with the image. Fig. 2 c shows the query being transformed with respect to the model for maximum overlap. By overlap we mean – given two points, a model point a and a query point b, the distance between a and b is below a certain threshold. It can be seen that six out of the seven pairs of points overlap – the pairs shown in Fig. 2 c are $(1, g), (2, c), (3, b), (4, d), (6, e), (7, f)$.

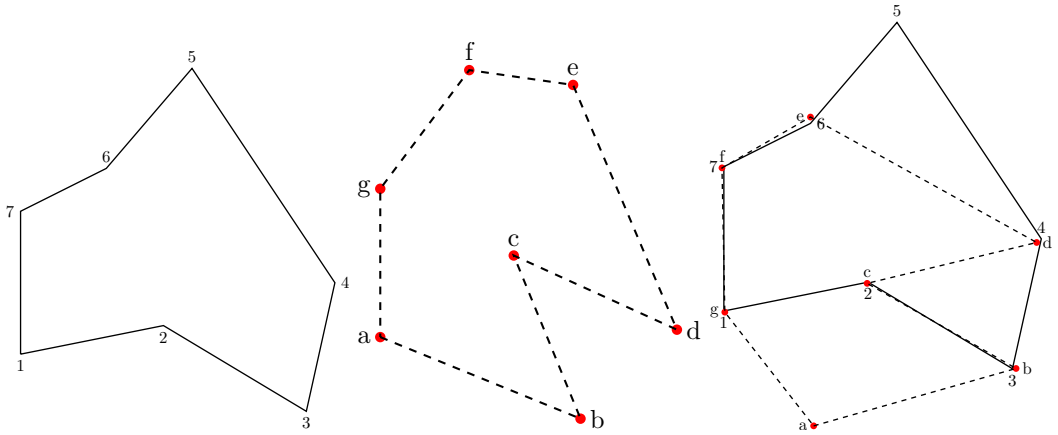
Reference Frames and Rigid Transformations

Our challenge becomes finding a 2-D *rigid transformation* (translation + rotation) that brings as many points as possible to within a tolerance distance. One easy way to create a 2-D rigid transformation is to select two points out of the set. Two points uniquely define a rigid transformation. These points are the *basis* to the transformation. Such a transformation is also called a *reference frame*, since it can be used to describe the origin and the principal directions, and the rest of the system can be positioned with respect to it. In other words, you can imagine yourself positioned at the origin of the system, so it is your "point of view" with respect to the rest of the world. You can rotate and translate every object in the world to coincide with the frame of reference.



(a) (b)

Figure 1: An example of an image (right) containing a partially occluded pair of scissors. Left: An image of a pair of scissors we use to recognize on the right image



(a) (b) (c)

Figure 2: (a)A seven point 2D model (b) A query (c) A transformation that co-incides six out of seven points from the query and model.

Looking at our example, we select two points, $a = (a_x, a_y)$ and $b = (b_x, b_y)$ build the reference frame as follows:

- The origin lies on point a, so the translation vector is the coordinates of a.
- The x direction lies on the line between a and b

This is all we need to determine the reference frame in 2-D! The theoretical basis is further discussed in Ch. **??**, so we will summarize it here: the x axis is a unit vector obtained by calculating the vector $c = b - a = (b_x - a_x, b_y - a_y$. Let us denote the components of c as (c_x, c_y). To normalize, divide each of c_x and c_y by the magnitude $\sqrt{({c_x}^2 + {c_y}^2)}$. The result is a unit vector pointing in the direction of the x axis of the reference frame. In a right-handed coordinate system the y axis is perpendicular to the x axis at the counter-clockwise direction. We can calculate its value using the fact that the two vectors constituting a 2D rotation matrix must be of the form:

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}$$

The first column is the x axis which we already know, so we can extract the magnitude of θ as $arcsin(x_2)$ or $arccos(x_1)$. By convention we will define θ in the range of $[-180, 180]^\circ$ or $[-\pi, \pi]$ in radians. The sign of θ is positive if x_2 is positive, negative if x_2 is negative and 0 if $x_2 = 0$. Based on that we can calculate y. However, we do not even have to do that, since a 2D rotation is specified by one parameter, the rotation angle θ, which we already know. Once we obtain the transformation we apply it to all the model points.

Let us apply this method to two points in our example. In Figure 4 (a) shows the example shape above on an axis system. as the basis for our reference frame. Let us select the points labeled 1 and 3 and call the frame $(1, 3)$ Their coordinates are $(3, 5)$ and $(23, 1)$, respectively.

Translation: Based on the formula above, point a defines the origin. It is currently at $(3, 5)$. Therefore, the points have to be translated by $(-3, -5)$ so that a lies on the origin of the new reference frame.

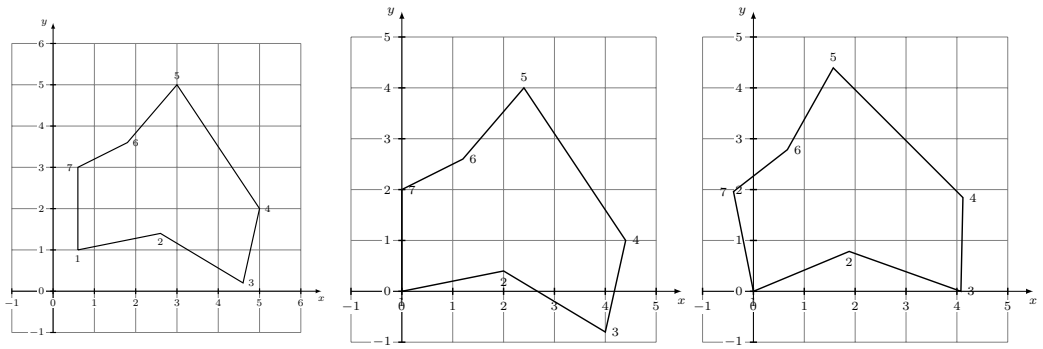
Rotation: The direction of the x axis is parallel to the vector $||b-a||$. The direction of the vector is $(b-a) = (20, -4)$. To normalize the vector, divide it by its magnitude which is $\sqrt{20^2 + 4^2} = \sqrt{416} = 20.4$. The normalized vector is $(0.981, -0.196)$. The rotation angle θ is $arcsin(-0.196) = -11.3^\circ$. Notice that this is the angle the vector between points 1 and 3 makes with the x axis. In order to align this vector with the x axis, we have to rotate the shape in Figure 4 (a) by +11.3 degrees to "rotate it back" into the x axis.

To obtain the shape in Figure 4 (b) we perform the translation, so that point 1 coincides with the origin. To obtain the transformed shape in 4 (c) we rotate the translated shape by 11.3 degrees.

Question: Given a model with n points, how many unique reference frames can we build?

Answer: The maximum number of *ordered* pairs is $n * (n - 1)$. Notice that in the example above we selected point 1 first and point 3 second. If we reverse the order, we will get a different transformation.

To verify that your transformation is right, notice that the two basis points have to lie on the x axis. Therefore, the coordinates of the first point in the basis have to be $(0, 0)$ and the coordinates of the second point have to be $(d, 0)$ where d is the Euclidean distance between the two points, in this case 20.4.



(a) (b) (c)

Figure 3: (a) The model from Fig. 2 (a). (b) The model translated so that the point labeled 1 is at the origin. (c) The model rotated so that points 1 and 3 coincide with the x axis.

Ranking a Reference Frame

To compare the model and query systems, we apply a transformation to the query, as shown in Fig. 2 b. Let us select points g and b as our reference frame, calling the frame (g, b). The original coordinates for the two points in Fig. 2 b. are $(10, 20)$ for g and $(23.5, 4.5)$ for b. The translation vector is therefore $(-10, -20)$. The direction of the x axis is $(13.5, -15.5)$. The magnitude is 22.55. The normalized vector is $(0.657, -0.754)$. The rotation angle is $-48.9°$. The result is shown in Fig. 4(b). As seen, five pairs of points coincide out of the possible seven – $(1, g), (2, c), (3, b), (4, d), (7, f)$. This is not a bad result, be as seen above, there is a transformation that overlaps six pairs. Which leads us the the following question:

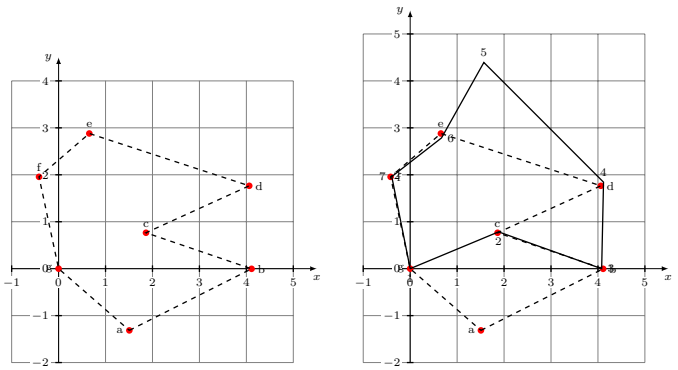
Question: Given a model with n points and a query with m points, how many unique transformations should we compare?

Answer: The maximum number of transformations $n * (n - 1)$ for the model and $m * (m - 1)$ for the query. Comparing all against all results in $n * (n - 1) * m * (m - 1)$ possible transformations. Notice, however, that redundancies may happen, For example, let (a_i, a_k) and (b_j, b_l) be selected as base pairs from model a and query b, respectively. Say (a_r, b_u) and (a_s, b_v) both coincide, then it is likely that similar coincidence sets will be found if (a_r, a_s) and (b_u, b_v) are selected as base pairs.

Hashing Transformations

Now that we have a way to transform the model and query shapes, we can utilize the power of hashing to conduct an efficient search for the transformation(s) that maximize the number of coinciding points. We construct a hash table that stores the geometric locations of the transformed points (hence the name Geometric Hashing). This allows us to simultaneously compare a query frame system to all the model frame systems. Geometric hashing has two stages:

Preprocessing: In this stage we build a hash table H, which has a bin for each cell in the frame system. The dimension of H is determined by the number of points that define a frame. Therefore, in our example, H will be two dimensional. The number of bins in H is the same as the number of cells in the frame system, so you may think of H as a two dimensional coordinate system divided



(a) (b)

Figure 4: (a) The query from Fig. 2 (b), transformed so point g is at the origin and point g and b coincide with the x axis. (b) The transformed model from Fig. 4(c) and query overlapping. Five points coincide.

into a grid at a specified resolution. If there is a point in the cell (p, q) in the frame system with basis (a_i, a_k), then (a_i, a_k) is placed in the bin $H(p, q)$. Every entry in H contains the frame identifier and the point identifier. Figure 5 shows H after two frames were inserted – frame $(3, 5)$ and frame $(1, 3)$. Each cell (p, q) contains the points whose x coordinates fall within the range $[p, p + 1)$ and whose y coordinates fall within the range $[q, q+1)$. We calculate the reference frame in a similar way for each model basis and transform the model in each reference frame, inserting the transformed copies into the table. After this stage, H contains multiple transformed copies of the model. H may take up a lot of space, but it saves a lot of time since the next stage, the recognition, is able to locate all the matches from all the reference frames at the same time. The preprocessing stage can be done offline and H can be used later in case we want to conduct multiple queries.

Recognition: In this stage the coordinates of the query are calculated according to some basis. The query points are then used as indices to H. Each query point is indexed into a cell containing transformed model points with similar coordinates. For each cell being index, a "vote" is given to the basis pair(s) with points found in the cell. The number of votes for a model basis pair is the number of coinciding points to the query (using the specified query basis pair). In the example in Fig. 6 the query points are marked in blue. As seen, model $(1, 3)$ got five votes and model $(3, 5)$ received two votes. In the end, the basis with the most votes is output.

Reference Frame in 3D

Reference frames in 3D are rather similar in principle to the 3D example. However, as explained in our previous discussions, rotations in 3D more than just a simple extension of 2D. To uniquely define a reference frame or a rigid transformation (translation + rotation in 3D), we need an ordered triplet of non-collinear points, a, b, c. The reference frame can be defined as follows (other ways are possible, but this one resembles the way internal coordinates are calculated):

1. The origin lies on point a.

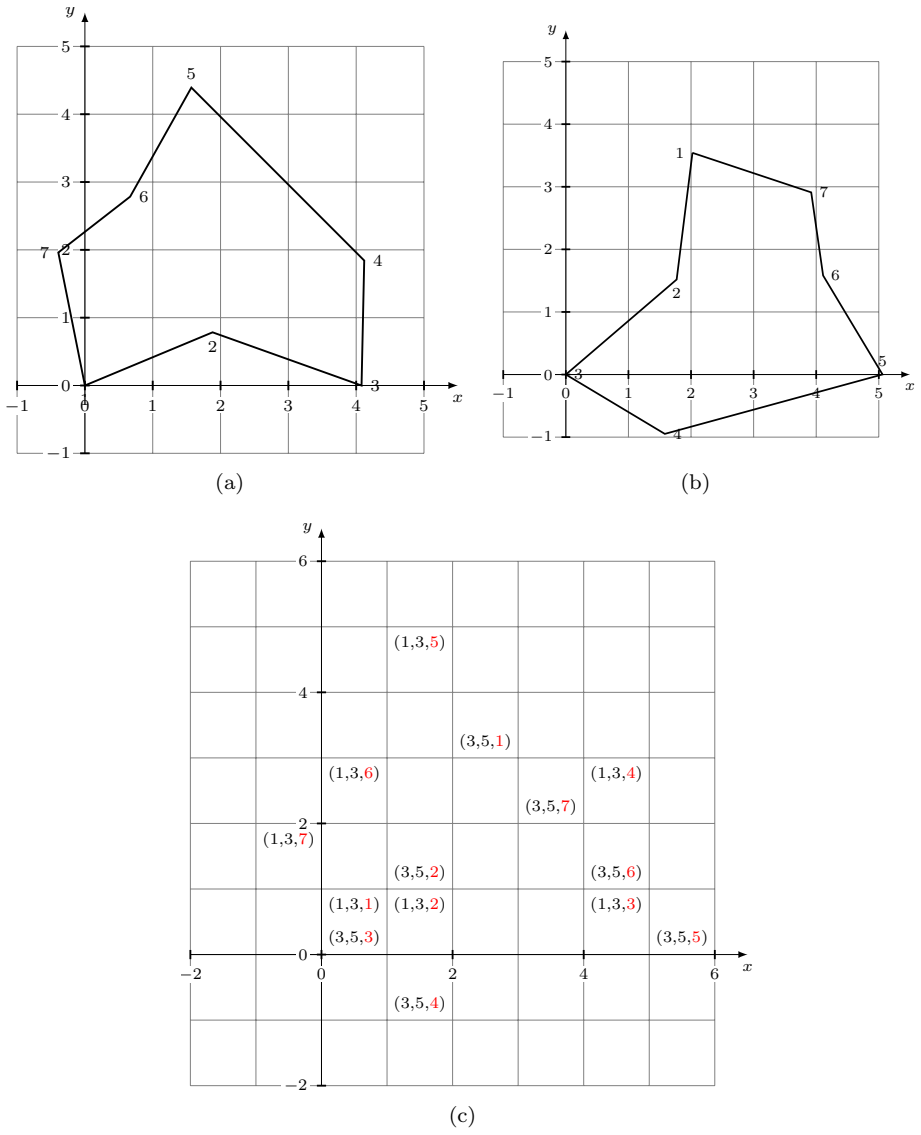


Figure 5: (a) The model transformed according to reference frame $(1, 3)$. (b) The model transformed according to reference frame $(3, 5)$. (c) The 2D Hash table containing the geometric locations of the two sets of points in (a) and (b). Each point is indexed according to its reference frame and point number.

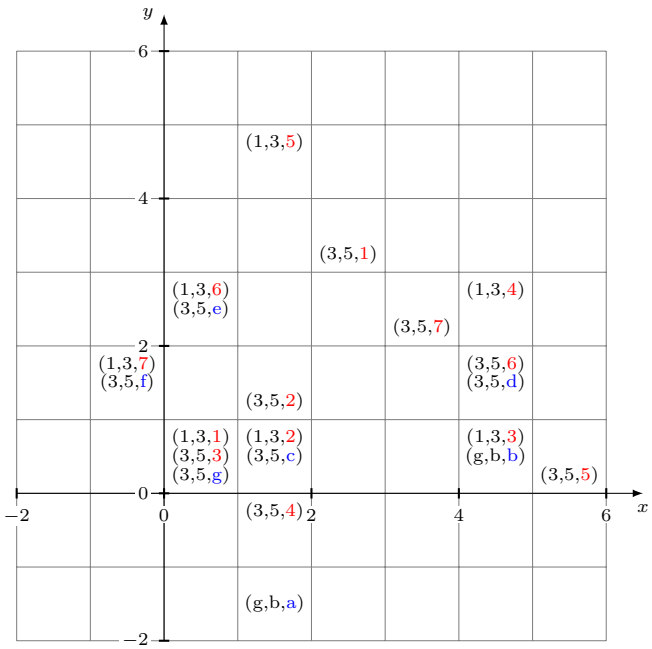


Figure 6: The table from Figure 5(a) with the query points used to recognize matching model points. The query points are indexed by their coordinates in a similar way to the model points, with the point name marked in blue.

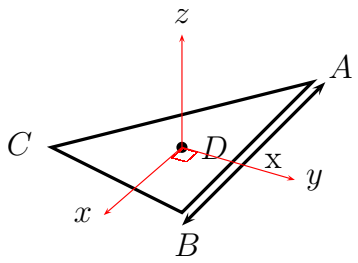


Figure 7: Constructing a 3D reference frame from three non-collinear points.

2. The direction of the x axis is the line between a and b. The x axis is therefore $||b-a||$, the normalized vector between the two points.

3. The xy plane is the plane defined by the three points a, b, c. To calculate the z axis, calculate the cross product between $b-a$ and $c-b$. The result is a vector perpendicular to the xy plane.

4. Now that we have x and z, the y axis is simply $x \times z$ (the cross product of x and z).

Notice that in this representation, the z coordinate value of a, b, c has to be zero. The transformation is illustrated in Fig. 7.

The side lengths are invariant to rigid transformation (rotation + translation) and can be used as the key to store the information in a hash table. In this case the hash table is a 3D grid, where each grid cube contains the points whose coordinates fall within the cube, indexed by their reference frame triplet.

Geometric Hashing and Protein Structure Analysis

Protein structures are 3D objects. Hence, we can use three non co-linear amino acids to define a reference frame in 3D. For example, we can use the $C-\alpha$ coordinate of each amino acid as our basis. Figure 8 shows an example of triplets of $C-\alpha$ atoms selected from a protein structure and put in a 3D grid. The reference frame can be calculated based on the formula above. In principle, every triplet of non-collinear $C-\alpha$ atoms can be used to create a reference frame, but it is possible to restrict the number of triplets by selecting points whose distances are within a certain distance values, usually an annulus defined by minimum and maximum radii. Typical values can be between 8 and 15Å . The reason is that residues that are very close to one another may not be informative with respect to the rest of the protein, and residues that are very far from one another may not give a lot of information about local matches.

Preprocessing: Once we define a reference frame, we compute the positions of the other amino acids in the frame and store the coordinates in a 3D hash table, tagged by their indices and reference frame number. We can do this for every possible triple of $C-\alpha$ atoms of the model in the preprocessing phase. However, this may make the hash table very large and may slow down computation, since this results in $O(N^3)$ triplets, where N is the number of amino acids, which can be in the hundreds. In general, let us define n as the number of points in our object, and R

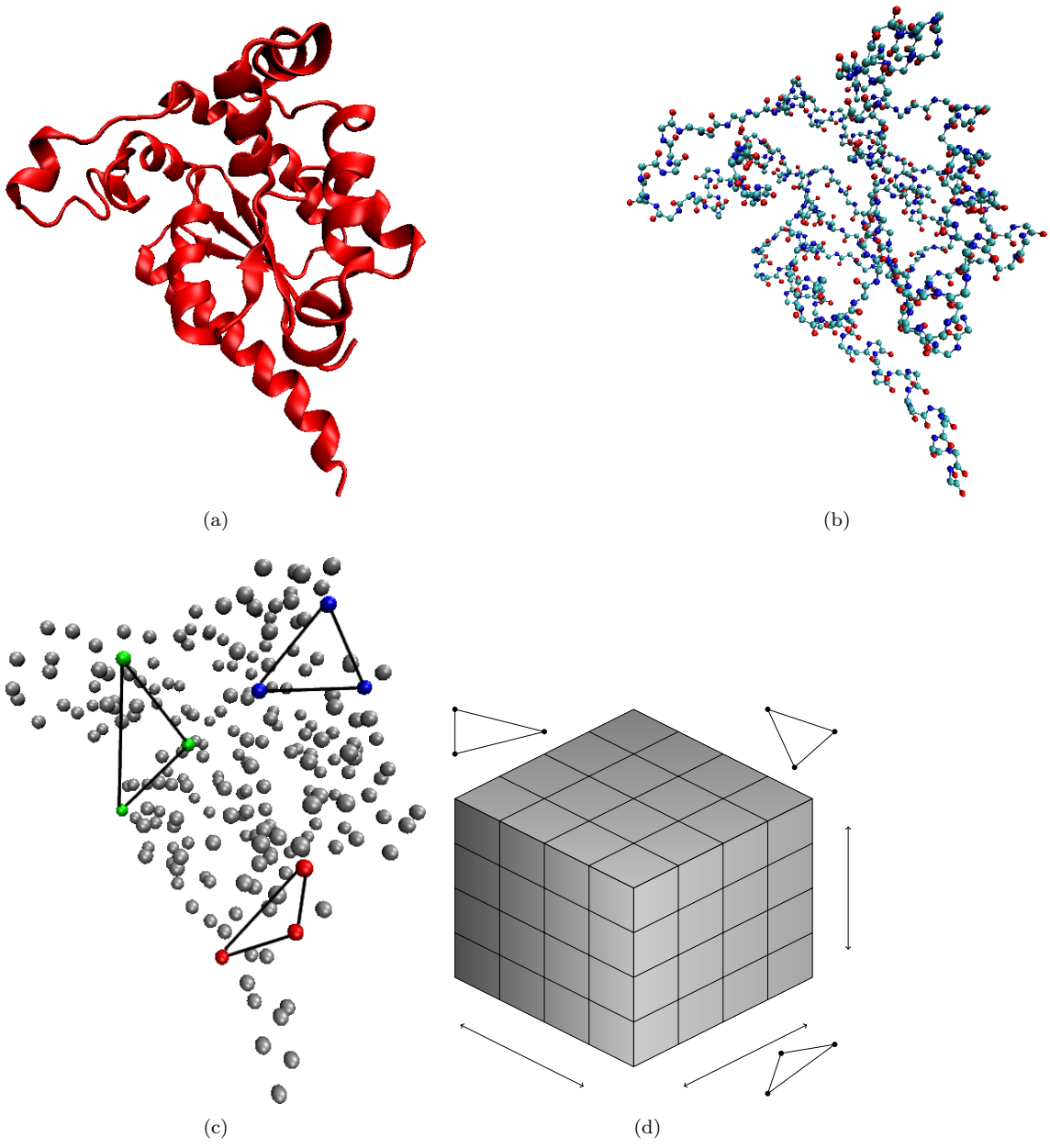


Figure 8: (a) A protein structure. (b) The ball-and-stick representation. (c) Triplets of $C-\alpha$ atoms selected as a reference frame.

as then number of reference frames. If s is the size of Hash entry, then the preprocessing stage takes $O(R \times n)$, since we have to transform every point according to R frames. The runtime for the recognition stage is at most $O(R \times n \times s)$. It should be noted that the pre-processing can be done once, offline, for multiple recognition queries.

Recognition: In the recognition stage we search the hash table with query points to find initial matches – that is, pairs of residues that overlap, just like in the 2D case. We do the following: First, pick a reference frame satisfying pre-specified constraints. Then, compute the coordinates of all other points in the current reference frame. We then use each coordinate to access the hash table to retrieve all the records. Each record contains information about a point – its amino acid number and reference frame number at the minimum. This information is called the "signature" of an entry. Then we "vote" for each reference frame. Every reference frame gets a "vote" for every match retrieved from the hash table. Reference frames with a high number of votes account for many matching points between the query and model. These high voting reference frames can be further improved by augmentation.

Augmentation: Potential matches retrieved by the recognition stage above can often be augmented by performing local optimization: For every match list containing residues from the model and the query, The match can then be augmented by calculating the alignment between the pairs that minimizes the RMSD (least RMSD or lRMSD). The alignment can potentially discover new matches between model and query residues. These new matches are pairs of residues that are now closer to one another than a pre-specified threshold. We stop either when the process converges (calculating the lRMSD alignment does not augment the match list) or when the lRMSD is greater than some threshold value ϵ. In many cases augmentation helps increasing the size of the match by finding nearby residues that would not be found by geometric hashing since they fall outside the search criteria using the original transformation. Distance measurements and the lRMSD algorithm are covered in Chapter **??**.

Motif Finding

LabelHash, an algorithm for matching substructural motifs to large collections of protein structures, uses hashing. The algorithm consists of two phases. In the first phase the proteins are preprocessed in a fashion that allows for instant lookup of partial matches to any motif. In the second phase, partial matches for a given motif are expanded to complete matches. The structural motif is defined by the backbone $C\alpha$ coordinates of a number of residues and (optionally) allowed residue substitutions for each motif residue which are encoded as labels.

The preprocessing stage has to be performed only once for a given set of targets; any motif can then be matched against the same preprocessed data. The targets can consist of single chains, but it is also possible to treat entire domains as a single target. This is useful for motifs that span multiple chains. During the preprocessing stage the aim is to find possible candidate partial matches. This is done by finding all n-tuples of residues that satisfy certain geometric constraints. The n-tuples are referred to as reference sets. Typically, n is small, say 3-tuples. All valid reference sets for all targets are stored in a hash table. In this case, each key is a sorted n-tuple of residue labels, and the value consists of a list of reference sets that contain residues with these labels in any order. In contrast to geometric hashing, LabelHash does not store transformed copies of the

targets for each reference set, which allows to store many more reference sets in the same amount of memory.

For a given motif of size m the LabelHash algorithm can look up all matches to a submotif of fixed size $n \leq m$, and expand each partial match to a complete match using a depth-first search. The partial match expansion is a variant of the match augmentation algorithm that consists of the following steps:

- Calculate the residue label correspondence between a motif reference set and the matching reference sets stored in the LabelHash table. (If more than one correspondence exists, all of them are considered.)
- Next, the match is augmented one residue at a time, each time updating the optimal alignment that minimizes the lRMSD. If a partial match has an lRMSD greater than some threshold ϵ, it is rejected. For a given motif point, all residues in a target that are within some threshold distance (after alignment) are retrieved. This threshold is for simplicity usually also set to ϵ. The threshold is set to be sufficiently large so that no interesting matches are missed.

The algorithm recursively augments each partial match with the addition of each candidate target residue. The residues added to a match during match augmentation are not subject to the geometric constraints of reference sets. In other words, residues that are not part of a reference set are allowed to be further from each other and more deeply buried in the core. For small-size reference sets, the requirement that a motif contains at least one reference set is therefore only a very mild constraint. As we will see in the next section, our approach is still highly sensitive and specific.

For a given motif, all the valid reference sets are generated. Any of these reference sets can be used as a starting point for matching. However, those reference sets that have the smallest number of matching reference sets in the LabelHash table may be more indicative of a unique function. Reference sets with a large number of matches are more likely to be common structural elements or due to chance. We could exhaustively try all possible reference sets, but for efficiency reasons we only process a fixed number of least common reference sets. Note that the selection of reference sets as seed matches is based only on frequency. Because of the preprocessing stage it now becomes feasible to expand matches from many different reference sets. The information stored inside a LabelHash file is stored so that only the relevant parts of the file need to be read from disk during matching.

Further Reading

- Geometric Hashing Lamdan, Wolfson 1988
- Molecular biology adaptation – Wolfson and Nussinov, 1989.
- For Patchdock see Duhovny et al., Lecture Notes in Computer Science 2452, pp. 185-200, Springer Verlag, 2002