Protein Structure Comparison and Structure Patterns - an Algorithmic Approach

Ingvar Eidhammer and Inge Jonassen
Dept. of Informatics,
Univ. of Bergen, Norway

[Ingvar.Eidhammer,Inge.Jonassen]@ii.uib.no

Outline

• Introduction and framework
• Description and representation
• Algorithms
  – Superposition
  – Alternating superposition and alignment
  – Double dynamic programming
  – Geometric hashing
  – Comparison by clustering
  – Multiple structure comparison
  – Local structure patterns
• Open problems and conclusions
### Organisation of tutorial

- **Part I**  Introduction  Inge
- **Part II**  Algorithms for pairwise structure comparison  Ingvar
- **Part III**  Comparison by clustering  Ingvar
- **Part IV**  Algorithms for multiple structure comparison and motif discovery, conclusions  Inge

### Part I - Introduction

- Motivation, motifs in protein analysis
- Protein comparison and classification
- Terminology
- Framework
  - structure description
  - alignment/equivalence
  - scoring
  - algorithm
- Example method
  - Alternating superposition and alignment
Motifs in Protein Analysis

Sequence $\rightarrow$ structure motifs
Sequence $\rightarrow$ function motifs
Structure $\rightarrow$ function motifs

Sequence structure/function motifs

• Pattern shared by a the sequences of a set of proteins with similar functions and/or structures.
• Match to pattern suggests structure/function for protein.

• Databases:
  – PFAM, PRINTS, PROSITE, PRODOM, BLOCKS, SMART, …

• Integrated database:
  – InterPro
Example sequence motif

```plaintext
xxx
V x
x x
x x
x x
C H
x \ / x
x Zn x
x / \ x
C H

C-x(2,4) - C-x(3) - [ILVMFYWC]-x(8)-H-x(3,5)-H
```

Sequence alignment

Alignment of chromo domains

[Sequence alignment diagram]
Motif Discovery

Unaligned Sequences/structures

Align

Pattern Discovery Method

Analyze alignment

Motif

Protein structure-function motifs

• PROCAT
  – A database of 3D enzyme active site templates
  – Thornton et al

• Fuzzy Functional Forms
  – structural descriptions of functional sites
  – www.geneformatics.com
  – Skolnick and Fetrow, 2000
Protein Classifications

- SCOP
  - Murzin et al
  - manually made
- CATH
  - Orengo et al
  - partly automated
- FSSP
  - Fold classification based on Structure-Structure alignment of Proteins
  - Fully automatic
  - Holm and Sander

Cartoon illustrating the three highest levels in CATH

Why compare structures?

- Learn about
  - structure-function relationships
  - evolution
  - common building blocks - motifs
- Make order out of the universe of protein structures
- Help structure prediction
Why is there more than one way?

- Proteins are complex three-dimensional structures.
- They can be described and compared in different ways
  - at different levels with
  - different constraints
- Choice depends on purpose of comparison
  - similarity of secondary structure packing
  - similar topology

- Example structure: 2nad

Some terminology

- A protein structure consists of one or more polypeptide chains.
- The order of amino acids along one chain is its primary structure (and its amino acid sequence).
- Some regions form regular local structure
  - alpha helices
  - beta strands
  - collectively called secondary structure elements (SSEs)
- Regions connecting SSEs are loops.
- Description of the type and location of SSEs is a chain’s secondary structure.
Terminology (cont’d)

• Three-dimensional coordinates of the atoms of a chain is its tertiary structure.
• Quarternary structure: describes the spatial packing of several folded polypeptides
• From CATH:
  – Class - composition of SSEs
  – Architecture - spatial packing of SSEs
  – Topology - includes arrangement of SSEs along the chain
  – (Homology - proteins with common ancestry apparent from sequence)

How to compare structures?

• Align
  – require co-linearity of paired-up elements
  – analogous to sequence alignment, but more complex

• Find similar substructures
  – ignore sequence order/topology

• Compare composition in terms of SSEs
Structure Motifs

- Common arrangements of SSEs (core motifs)
  - beta barrel

- Binding/active site (residues)
  - catalytic triad,
  - zinc binding residues

Framework

Diagram showing the process of Structure Motifs with steps labeled (1) and (2).
ISMB Tutorial 2001
Eidhammer and Jonassen

Structure 1

Structure 2

Feature extraction

Structure description 1

Structure description 2

Comparison Algorithm

Constraints

Scoring

RMSD

Score

Equivalence

Assessment

Statistical significance

Accurate?
Structure Description
- Levels

- Atom/atom group
- Residue
- Fragment
- Secondary structure element (SSE)
- Structure described by elements of the chosen type

Structure Description
- Features

- Geometry/architecture
  - coordinates/relative positions, …
- Topology
  - sequential order of residues along backbone, ...
- Properties
  - physio-chemical properties of residues, …
Equivalences and Alignments

- Object $A$ with elements $a_1, a_2, \ldots, a_m$
- Object $B$ with elements $b_1, b_2, \ldots, b_n$

**Equivalence:** $L(A,B) = (a_{i1}, b_{j1}), (a_{i2}, b_{j2}), \ldots, (a_{il}, b_{jl})$

**Alignment:** co-linear equivalence
- $i_1 < i_2 < \ldots < i_l$ and $j_1 < j_2 < \ldots < j_l$

- Can be extended to multiple structures
Scoring Equivalences

• Coordinate based
  – defined using a transformation of one structure onto the other:
    root mean square deviation - RMSd

Coordinate based RMSd

• Define transforms $T$ that consists of a
  – translation
  – rotation
• Structures $A$ and $B$ consist of coordinate sets $A_1...A_m$
  $B_1...B_m$ - equivalence $(A_1,B_1),..,(A_m,B_m)$

$$RMSd(A, B) = \min_T \sqrt{\sum_{i=1}^{m} (A_i - TB_i)^2}$$
Calculating coordinate based RMSd

- Need to find the transform that minimises the equation.
- Least square minimisation to find best combined transform (Muirhead et al 1967; Rao & Rossman 1973)
- Two step method:
  - Relocate origin to center of mass for A and B
  - Find best rotation:
    - Stepwise search of rotational space (Remington and Matthews, 1978)
    - Least Squares minimization (McLachlan, 1972)
    - Matrix methods
      - singular value decomposition (McLachlan, 1979)
      - Langrangian multipliers (Kabsch 1976, 1978)

Scoring Equivalences

- *Coordinate* based
  - defined using a transformation of one structure onto the other: root mean square deviation - RMSd
- Similarity of *properties* between equivalenced elements
  - conserved/similar amino acid
- Similarity of *relations* between pairs of equivalenced elements
  - similar distances, internal RMSd
Distance based RMSD

- No need to transform.
- Definition:

\[
RMSD_D(A, B) = \frac{1}{m} \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{m} (d_{ij}^A - d_{ij}^B)^2}
\]

where
- \(d_{ij}^A\) is the distance between atoms \(i\) and \(j\) in structure \(A\)

- Linear relationship between RMSD and RMSD\(_D\)
- RMSD\(_D\) cannot distinguish between mirror images.
Comparison Algorithms

- Compare two structure descriptions - find equivalence (alignment) having high score.
  - Computationally expensive

Dynamic Programming - DP

- **Sequences**
  - $S, T$ can be aligned optimally in time $O(|S||T|)$

- Next slides gives a summary of sequence alignment by DP.
Edit distance

- The edit distance between S1 and S2 is the minimum number of deletions, insertions and substitutions needed to transform S1 to S2.
- |S1|=m, |S2|=n
- D(i,j) is the edit distance between S1[1..i] and S2[1..j].

- Recurrence relation:
  - basis:
    
    \[
    \begin{align*}
    D(0,j) &= j \text{ for all } 0 \leq j \leq m \quad \text{Insert } j \text{ characters in } S1 \\
    D(i,0) &= i \text{ for all } 0 \leq i \leq n \quad \text{Delete } i \text{ characters from } S1
    \end{align*}
    \]

Edit distance (cont’d)

\[
D(i,j) = \min \begin{cases} 
D(i-1,j) + 1 & \text{Delete character } i \text{ from } S1 \\
D(i,j-1) + 1 & \text{Insert character in } S1 \\
D(i-1,j-1) + t(i,j) & \text{if } t(i,j) \neq 0, \text{ substitute character } S1(i) \text{ with } S2(j) \\
\end{cases}
\]

where t(i,j)=0 if S1(i)=S2(j) and t(i,j)=1 otherwise.

D(m,n) is the edit distance between S1 and S2.
Dynamic programming
- tabular computation

- Allocate table $D[0..m,0..n]$

- Fill in row 0 and column 0

  \[
  D(0, j) = j \text{ for all } 0 \leq j \leq m \\
  D(i, 0) = i \text{ for all } 0 \leq i \leq n
  \]

- Fill in remaining elements, for example row-wise or column-wise

  \[
  D(i, j) = \min \left\{ D(i-1, j) + 1, D(i, j-1) + 1, D(i-1, j-1) + t(i, j) \right\}
  \]

- $D(m,n)$ gives edit distance $S1,S2$.

- Find alignment/edit transcript by tracing back from $m,n$ to $0,0$

Alignment of biological sequences

- Use scoring matrix that rewards alignment of similar amino acids.

- Find alignment that maximises similarity of aligned amino acids.

- Gaps penalised typically using affine gap penalty

  - gap of length $k$ penalised by $ak+b$

- Simple modification of edit distance algorithm
Global alignment of two sequences

- Assuming linear gap penalty ($b=0$):

\[
S(0, j) = -ja \text{ for all } 0 \leq j \leq m
\]
\[
S(i,0) = -ia \text{ for all } 0 \leq i \leq n
\]
\[
S(i,j) = \max \left\{ S(i-1,j) - a, S(i,j-1) - a, S(i-1,j-1) + \text{score}(i,j) \right\}
\]

Example of pairwise global alignment

<table>
<thead>
<tr>
<th>I</th>
<th>N</th>
<th>D</th>
<th>U</th>
<th>S</th>
<th>T</th>
<th>R</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>-0.5</td>
<td>-1.0</td>
<td>-1.5</td>
<td>-2.0</td>
<td>-2.5</td>
<td>-3.0</td>
<td>-3.5</td>
</tr>
<tr>
<td>-0.5</td>
<td>1.0</td>
<td>0.5</td>
<td>0.0</td>
<td>-0.5</td>
<td>-1.0</td>
<td>-1.5</td>
<td>-2.0</td>
</tr>
<tr>
<td>-1.0</td>
<td>0.5</td>
<td>2.0</td>
<td>1.5</td>
<td>1.0</td>
<td>0.5</td>
<td>0.0</td>
<td>-0.5</td>
</tr>
<tr>
<td>-1.5</td>
<td>0.0</td>
<td>1.5</td>
<td>1.0</td>
<td>1.0</td>
<td>0.5</td>
<td>1.5</td>
<td>1.0</td>
</tr>
<tr>
<td>-2.0</td>
<td>-0.5</td>
<td>1.0</td>
<td>1.0</td>
<td>0.5</td>
<td>0.5</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>-2.5</td>
<td>-1.0</td>
<td>0.5</td>
<td>0.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.5</td>
<td>2.0</td>
</tr>
<tr>
<td>-3.0</td>
<td>-1.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.5</td>
</tr>
<tr>
<td>-3.5</td>
<td>-2.0</td>
<td>0.5</td>
<td>-0.5</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.5</td>
</tr>
<tr>
<td>-4.0</td>
<td>-2.5</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-2.0</td>
<td>-1.5</td>
</tr>
</tbody>
</table>

Score:
- Identical letters: 1.0
- Different letters: -0.5
- Gap: -0.5
Local Alignment

- Sequences: $A$ and $B$
- Pick
  - substring $a$ from $A$
  - substring $b$ from $B$
- so that $a$ and $b$ can be aligned giving score $Sc(a,b)$
- and $Sc(a,b)$ is maximum over all substrings $a$, $b$ from $A$, $B$. 
Local alignment - recurrence relation

\[ S(0, j) = 0 \text{ for all } 0 \leq j \leq m \]
\[ S(i,0) = 0 \text{ for all } 0 \leq i \leq n \]
\[ S(i,j) = \max \begin{cases} 
S(i-1,j) - a \\
S(i,j-1) - a \\
S(i-1,j-1) + \text{score}(i,j) \\
0
\end{cases} \]

Dynamic Programming for alignment of structures

- **Structures:**
  Scoring of equivalenced elements are not independent.
  - Coordinates transformed - rigid body
  - Relations between elements (e.g., distances)

- For DP to work, the optimality of earlier made choices cannot be affected by new choices.
  - Does not hold for structure alignment
Alternating superposition and alignment

Align $A$ and $B$.
1. $P$= Initial alignment/equivalence
2. Superpose $A$ onto $B$ based on $P$ → make scoring matrix $R$
3. Align $A$ and $B$ using DP and $R$ → new $P'$
4. If ($P'\neq P$), then $P=P'$, go to 2

A simple alternating method
- initial alignment

- Multiple ways
  - user specifies list of residue pairs
  - motif matches

- Given alignment (list of residue pairs), superpose structures A and B to minimize RMSd

A simple alternating method
- from superposition to alignment

- From Superposition to scoring matrix and alignment
  - let $d(A_i, B_j)$ be the Euclidean distance between $A_i$ and $B_j$ after superposition
  - let $\tau$ be the upper limit on distance between residues to be rewarded
  - $R$ is a scoring matrix that can be used for DP alignment of the superposed structures

$$R(A_i, B_j) = \frac{1}{d(A_i, B_j)} - \frac{1}{\tau}$$

- zero gap penalty, and allow zero diagonal step
A simple alternating method -
dynamic programming

\[ D(0, j) = 0 \text{ for all } 0 \leq j \leq m \]
\[ D(i,0) = 0 \text{ for all } 0 \leq i \leq n \]
\[ D(i,j) = \max \left\{ D(i-1,j), D(i,j-1), D(i-1,j-1) + R(i,j) \right\} \]

A simple alternating method -
dynamic programming (cont’d)

- **Forward:**
  - fill in matrix row-wise/column wise
  - \( D(m,n) \) gives total score of best possible alignment (set of co-linear residue pairs)

- **Traceback:**
  - find all contributing (non-zero) diagonal moves
  - this is the new alignment to be used for the next superposition
Part II - Algorithms for pairwise structure comparison

- Double dynamic programming

- Geometric hashing

Double Dynamic Programming

Double dynamic programming

- Ideally, one wishes to simultaneously align and superpose the structures
- Thus the independency requirement is violated
- However, several heuristic algorithms trying to extend DP to solve this problem have been developed
- One such method is SSAP (Structure Sequence Alignment Program), based on Double Dynamic Programming (DDP)

DDP - outline

- DDP ideas
- Algorithms
- Iterated DDP
Ideas for DDP

- Use two levels of dynamic programming, to construct a *high level* scoring matrix that can be used for ordinary DP

- For each residue pair \((a_i, b_j)\) this matrix should show how likely it is that the pair is on an optimal alignment

- For each \((a_i, b_j)\), this likelihood is found by a *low level* optimal alignment with the constraint that \((a_i, b_j)\) is part of the alignment

- The scores along the low level alignments are accumulated in the high level scoring matrix
Ideas for DDP (cont’d)

• A low level scoring matrix $ijR$ is defined for each pair $(a_i,b_j)$

• $ijR_{kl}$ is a score showing how well the residue $a_k$ fits to $b_l$ under the constraint that $a_i$ is aligned to $b_j$.

• Low level DP is performed using $ijR$ (in principle for each pair $(i,j)$)

• The contribution from $ijR$ is all $ijR_{kl}$ such that $(a_k,b_l)$ lies on the optimal (low level) path

Algorithm for DDP

$R := \{0\}$  \hspace{1cm} High level scoring matrix 0

for each pair $(a_i,b_j)$ do

compute the low level scoring matrix $ijR$

$(s,P) := DP_{ijR}(A,B)$  \hspace{1cm} Low level DP forced through $(a_i,b_j)$

$P$ is the optimal path, $s$ the score

forall $(a_p,b_q)$ in $P$ do $R_{pq} := R_{pq} + ijR_{pq}$  \hspace{1cm} Accumulate into $R$

end

$(s,P) := DP_R(A,B)$  \hspace{1cm} High level DP using $R$
Noise

- As $R$ contains a sum of values from the low level matrices, it is normalised before high level DP

- $R$ is the sum of $nm$ low level matrices, most of them representing pairs which are not in the final alignment, thus introducing noise

- Therefore, only low level alignments scoring above a cutoff is accumulated
The low level scoring matrices

- \(|R_{kl}|^2\) should be based on superposition bond on \(a_i\) and \(b_j\)

- Define local reference systems around \(a_i\) and \(b_j\) e.g. by using (the \(C_{\alpha}\)s of) \(a_{i-1}, a_i, a_{i+1}\) and \(b_{j-1}, b_j, b_{j+1}\)

- The coordinates of \(A\)'s and \(B\)'s residues are transformed into the respective coordinate systems

- One simple scoring scheme for aligning \(a_k\) and \(b_l\) depends only on the distance between \(a_k\) and \(b_l\) in their respective coordinate systems

Example

Structure A

Coordinate system coincidence at \(a_i, b_j\)
More advanced scoring scheme

- A direction component, realised as a function of the vector $d_{kl}$
- An orientation component, reflecting the difference in reference frames around $a_k$ and $b_l$ (after superposition)
- A sequence component $g_{kl}$, calculated as an increasing function of $|k-i| + |l-j|$. This should damp the contribution from near neighbours in the sequence
- A spatial component $h_{kl}$, calculated as a decreasing function of $d_{ik} + d_{jk}$, where $d_{ik}$ is the distance between $a_i$ and $a_k$. This will damp the contribution from large distances in space (want high contribution from pairs near in space and far in sequence)

Iterated DDP

- As DDP is heuristic, one might achieve better results by iterating, doing a DDP in each cycle
- This, together with limiting the work for each DDP is developed into a program called SAP (Taylor 1999)
Iterated DDP (cont’d)

- In each cycle, only some of the pairs \((a_i, b_j)\) are used for low level DP

- Some pairs, called seeds, are chosen for the first cycle either chosen randomly, or found by a (motif discovery) program (e.g. SPratt)

- In each cycle the high level scoring matrix \(R\) is updated, and also the selected pairs \((a_i, b_j)\) for the next cycle

- A bias matrix \((Q)\) is used in updating \(R\)

Algorithm for iterated DDP

initialise the bias matrix \(Q\)
select the seed pairs using \(Q\)

\textbf{iter}

place the selected pairs in \(I\)
\(R := \{0\}\) \hspace{1cm} Set high level matrix to zero

\textbf{for} each pair \((a_i, b_j)\) in \(I\) \textbf{do}

\((s,P) := \text{DP}_{sR}(A,B)\) \hspace{1cm} Low level DP

accumulate the low level results to \(R\)

\textbf{end}

update \(Q\) using old \(Q\) and \(R\)

\((s,P) := \text{DP}_Q(A,B)\) \hspace{1cm} High level DP

select new pairs \((a_i, b_j)\) based on \(Q\)

\textbf{until} termination criterion is satisfied
Similarity Between Alternating Approach and Iterated DDP

Algorithm statements to explain

- How is $Q$ initialised?
- How many residue pairs should be chosen in each cycle?
- How should $Q$ be updated?
- Should the high level scoring matrix contribute to the lower ones?
- What is the termination criterion?
Initialising the bias matrix $Q$

$Q$ is initialised by using the three components:

- Secondary structure
- Burial
- Amino acid similarity

Selecting pairs and updating $Q$

- The highest values in $Q$ are used for selecting the pairs for low level DP
- Continuity through the cycles ($p$) is maintained by using $Q$ as base for incremental revision(weakening contribution from initial $Q$): $p^{+1}Q := p^Q/2 + \log(1+p^{+1}R/20)$
- 10-20 pairs are initially selected, but the number is gradually increased, (and more ”true pairs” are found)
- The number of selected pairs is $K = 10 + \frac{p + 1}{20} \sqrt{mn}$
Coherence problem

- The initial selection of residues pairs might be quite random with respect to the final "true" equivalences

- As a consequence, the comparison of their environments might provide little coherent direction towards the final solution

- Although the bias matrix provides a platform from which the selection of pairs can be refined, it has no effect on the scores derived from the low-level matrices

High level contribution to low level

- A contribution from the bias matrix can therefore be introduced to provide stability into early cycles

- This is done by using a revised matrix

\[ R^*_{ij} = R_{ij} + pQG(p, I) \]

where p is the cycle number and G is the Gaussian transform function

- This provides a smooth transition from the, ideally, local information in the bias-matrix into the full global view provided by the comparison of the residue environments
Termination criterion

- The iteration should stop when the selected pairs coincide with the best (high level) path

- Let $U$ be the number of selected pairs on the best path using $Q$, and $V$ the number of selected pairs outside. Then stop if \( \frac{U}{U+V} \) becomes 1, or stops to increase

\[
\begin{array}{|c|c|c|c|c|c|c|c|c|}
\hline
& S & & & & & & & \\
\hline
& & & & & & & & \\
\hline
& & & & & & & & \\
\hline
& & & & & & & & \\
\hline
& & & & & & & & \\
\hline
& & & & & & & & \\
\hline
& & & & & & & & \\
\hline
\end{array}
\]

- An upper limit for the number of cycles must be defined

Geometric Hashing


Geometric Hashing - Outline

- Ideas
- Geometric hashing for 2 D
- Geometric hashing for structure comparison
- Geometric hashing for SSE-representation

Geometric hashing

- Geometric hashing was originally developed as a computer vision technique for matching geometric features
- It can be used to find common subfigures, invariant under rotation, translation and scale
- In structure comparison it can be used
  - as a full comparison method
  - for finding seeds for other methods (methods that use iteration or clustering)
- We will first describe Geometric Hashing for 2D figures
2D geometric hashing

- Two figures are given, a model $A$, and a query $B$, described by $m$ and $n$ points, respectively

- Find common subfigures, invariant under rotation and translation (scale is not used)

- One approach is to "place the query on top of the model", and consider how many points coincide (here ignoring the edges)

---

Example

Model

Query

Query over model for maximum coincidence of points (seems to be 6)
Coincidence sets

- Finding the maximal coincidence set is NP-hard
- In this context, we want to find *all* coincidence sets with the number of pairs over a given threshold

Reference frames

- Define coordinate systems for both figures \((A, B)\), called reference frames
- Two points (*basis pair*) can define a reference frame, e.g. origo at one of them, and one of the axis through both points
- The coordinates of the points are computed in the reference frame, constituting a reference frame system
- Count how many pair of points (one from each figure) have the same coordinates
Example Reference Frame Systems

Model, basis (1, 3)

Five coincident points
(1, g) coordinate (0,0)
(2, c) (3,2)
(3, b) (6,0)
(4, d) (8,4)
(7, f) (0,4)

Query, basis (g, b)

Two coincident points
(3, g) coordinate (0,0)
(2, c) (3,2)

Point 6 (2,6) not found coincident with point e (2,5)
Remarks

- The number of coincident points depends on the resolution of the coordinate system, on the basis pairs used.

- Generally, all combination of points should be used as basis pairs, resulting in comparing \[ \frac{m(m-1)}{2}, \frac{n(n-1)}{2} \] reference frame systems.

- Using all combinations might introduce redundancy. Let \((a_i, a_j)\) and \((b_j, b_i)\) be the basis pairs, and \((a_r, b_u)\) and \((a_s, b_v)\) both coincide. Then it is likely that the same coincidence set is found if \((a_r, a_s)\) and \((b_u, b_v)\) are used as basis. Note however that similarity and not exact equality is used.

Hashing

- For dealing efficiently with all combinations, hashing is used. It is especially efficient when several queries are to be compared to one model, or to several models.

- The comparison problem can be formulated as: given a query reference frame system, for each model reference frame system, in how many cells are there points from both the query and model frame system.

- The hashing technique makes it possible to simultaneously compare a query frame system to all model frame systems.
Preprocessing

- In this example, a 2D hash table $H$ is used. It has a bin for each cell in the frame systems. In a **preprocessing** phase, the coordinates of all points in each model frame system are found. 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 $H(p,q)$.

- Since all pairs of points from the model will (generally) act as basis pairs, totally

$$m \frac{m(m-1)}{2}$$

pairs will be in $H$.
Recognition

- The query is compared to the model in the recognition phase.

- A pair is chosen as basis, and the coordinates of the other points are calculated.

- These coordinates are used as indices into H, and for each cell being indexed, a vote is given for the (model) basis pairs 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).

Example Recognition

Query, basis (g,b)

Five votes for (1,3)
Two votes for (3,5)
Extensions

• Labels (e.g. colours and/or forms) assigned to the points might also be included, such that coinciding points also must have equal labels. This can be implemented in two ways:
  – Storing the labels in the table
  – The table can be hashed by using the labels in addition to the coordinates
• It is straightforward to extend the hashing technique to include several models, such that a query is simultaneously compared to several models. The only extension in the hash table is that a model identification must be stored along with the basis pairs

GH for structure comparison

• Since comparing structures should be invariant to translation and rotation, GH is a good candidate method, when the order of the residues (elements) along the chain is ignored

• Finding a coinciding set between two structures then means finding an equivalence (not necessarily an alignment)

• A 3D reference (frame) system must be defined, and often the $C_\alpha$-coordinates of three (non-colinear) residues are used
Algorithm for preprocessing

\begin{algorithm}
\begin{algorithmic}
\FOR{each model $M_u$}
\FOR{each (unordered, noncollinear) triples $(a_i, a_k, a_p) \in M_u$}
\STATE calculate the reference frame $R_{ikp}$
\FOR{each residue $r$}
\STATE calculate $F = F(a_i, a_k, a_p, a_r, p_L)$ \textit{;index in $H$}
\STATE $H(F) := H(F) \cup (M_u, R_{ikp})$
\ENDFOR
\ENDFOR
\ENDFOR
\end{algorithmic}
\end{algorithm}

The label $L$ is included in the hashing

Algorithm for recognition

\begin{algorithm}
\begin{algorithmic}
\REPEAT
\STATE initialise the vote table $V$ to 0
\STATE choose three atoms $(a_r, a_k, a_p)$ from the query as basis
\STATE calculate the reference frame system $R_{jls}$
\FOR{each residue $q$}
\STATE calculate $F = F(a_r, a_k, a_p, q_j, l)$
\FOR{each pair $(M, R) \in H(F)$}
\STATE $V(M, R) := V(M, R) + 1$
\ENDFOR
\ENDFOR
\UNTIL{satisfactory coincidence sets are found}\OR all query reference frames are used
\end{algorithmic}
\end{algorithm}
Remarks

- The result of the recognition is a list of \((M, R_M, R_B)\), showing that there is a coincidence set between the model \(M\) and the query \(B\), becoming evident when the reference frames \(R_M\) and \(R_B\) are used.
- The residues of the coincidence sets are known (or can be found), and a superposition can be done for checking the substructure similarities.

- Techniques for reducing the run time for Geometric Hashing exist.
- Techniques also exist for "practical adaption" (e.g., checking neighbouring cells).

Geometric hashing for SSE-representation

- Typically the SSE’s are represented as sticks.
- By use of two sticks, a coordinate reference frame can be defined (usually by placing one axis along one of the sticks).
- An entry in the hash table might be (Holm and Sander) a list where each list-element contains:
  - identification of the SSE
  - identification of the basis
  - the type of SSE (alpha, beta)
  - the midpoint of the SSE (in the reference frame)
  - the direction of the SSE
  - possible other information.
Part III - Comparison by Clustering

- Introduction
- Compatibility and consistency
- Clustering methods
- Clustering by use of transformation
- Clustering by use of relation
- Clustering as graph problem

Comparison by Clustering

- The clustering methods for structure comparison works by first finding small similar substructures in the two structures, and then constructing larger similar substructures by joining smaller ones
- Generally they allow for insertions, deletions and topological permutations

- The word *clustering* is a bit misleading, not all ”objects” have to be clustered. It is used here, since it is widely used in papers describing the relevant methods. (Perhaps *grouping* would be better)
Compatibility

• The basic objects are *compatible elements*, one from each structure

• Two elements are compatible if they share some similarities, what kind of similarities vary with the method
  – Residues may be compatible if they are the same amino acid, or in the same amino acid group
  – Secondary structures may be compatible if they are of the same type
  – For fragments may equal intern distances be required

Seed matches

• The methods first find *seed matches* between the structures, and then join these into larger clusters, a cluster representing one substructure from each structure
Consistency

- A seed match (being a basic cluster) consists of one or several pairs of compatible elements

- The pairs must be consistent, meaning that the two substructures consisting of the elements from the pairs together must be compatible, i.e. similar to a certain degree

- Seed matches are joined into consistent clusters, and scored, measuring how similar the two substructures are

Compatibility and consistency

- *Compatibility* is a binary relation between elements from different structures

- *Consistency* is a binary relation between clusters (of compatible elements)
Compatibility and consistency

Example

\[ (A_i, A_k) \text{ not } "\text{similar to}" (B_j, B_l) \]
The philosophy

- Find seed matches, i.e. sets of consistent pairs of compatible elements \( SM = \{ P_i \} = \{ (A_{i,j}, B_{i,j}) \} \)
  - All elements in a seed match must be different
  - An element may be in several seed matches

- Group consistent seed matches iteratively into clusters, representing the \((k)\) substructures with highest score

- Optional refinement (often using the iteratively alternating superposition/alignment)

The methods

The methods vary in how they solve the following:

- How is compatibility defined
- How to find the compatible elements, and the seed matches
- How is consistency defined
- How is clustering performed – clustering algorithms
- How are the clusters scored
- How is the refinement done (if at all)
Searching for seed matches

Different searching methods are used:

- Straightforward search, when each seed match consists of only one pair
- Geometric hashing, elements in seed matches constraint to lie within (small) spheres
- ...

Consistency

- Two clusters $C_1 = (C_1^A, C_1^B), C_2 = (C_2^A, C_2^B)$, (a cluster consisting of one or more seed matches) can be joined if they are consistent

- This means that the substructure of $A$ consisting of $\left( C_1^A \cup C_2^A \right)$ must be “similar” to the substructure of $B$ consisting of $\left( C_1^B \cup C_2^B \right)$
Decide consistency

To decide consistency, one can use:

- **transformation** ($\tau$) between compatible elements from different structures
- **relation** ($\rho$) between elements of the same structure

Consistency

\[ A_i \xleftarrow{\rho_{ik}} A_k \xrightarrow{\tau_{kl}} B_j \xleftarrow{\rho_{jl}} B_l \]

$A_i$ compatible with $B_j$
$A_k$ compatible with $B_l$
Is $(A_i, B_j)$ consistent with $(A_k, B_l)$?

Yes, if $(\tau_{ij} = \tau_{kl})$ or $(\rho_{ik} = \rho_{jl})$
Example transformation

\[ A_i \text{ compatible with } B_j \]
\[ A_k \text{ compatible with } B_l \]

\((A_i, B_j)\) is consistent with \((A_k, B_l)\) if
the transformation for best superposition of \((A_i, B_j)\)
is similar to the transformation for the best
superposition of \((A_k, B_l)\)

Example relation

\[ A_i \text{ compatible with } B_j \]
\[ A_k \text{ compatible with } B_l \]

\((A_i, B_j)\) is consistent with \((A_k, B_l)\) if
\[ d_1 \approx d_3 \land d_2 \approx d_4 \land \gamma_1 \approx \gamma_2 \]
The clustering step

\[ \begin{align*}
A_i & \quad A_j \quad A_k \\
B_h & \quad B_i & \quad B_k
\end{align*} \]

\[ \begin{align*}
A_i & \quad A_j \quad A_k \\
B_h & \quad B_i & \quad B_k
\end{align*} \]

Compatible

Consistent

Cluster \( C_1 \)

Consistent?

Cluster \( C_2 \)

Consistency test

- \( C_p = \{ P_1, P_2, P_3, \ldots \} \) \quad \( P_i = (A_{k_i}, B_{k_i}) \)
- \( C_q = \{ Q_1, Q_2, Q_3, \ldots \} \)
- Can \( C_p \) be joined with \( C_q \) \? (\( C_p \) consistent with \( C_q \)?)

- Use of local criteria: Use the pairs \( (P_p, Q_q) \) for test
  - Transitivity: \( \text{cons}(P_i, Q_j) \Rightarrow \forall k l : \text{cons}(P_k, Q_l) \)
    - Enough to test one pair (Example =)
  - Transitivity not satisfied
    - Must test each pair (in principle) (Example distance)

- Use global criteria
  - Test the clusters \( C_p \) \( C_q \) ”as units”
Clustering methods

- Linear clustering
- Parallel linear clustering
- Hierarchical clustering

Linear clustering

Seed matches $P_i = (A_i, B_i)$

- One current cluster
- Add one consistent seed match to current cluster
- The result may depend on the order of the seed matches (due to inaccuracy)

Algorithm

select one seed match as current cluster $C$

for each other seed match $sm$ do
  if $sm$ is consistent with $C$ then let $C$ be $sm$ joined to $C$
end
Parallel linear clustering

- Several current clusters

**Algorithm**

make a current cluster of each seed match (retaining them)

**repeat**

find the seed match \( sm \) ”most consistent” to a cluster \( C \)

**if** the consistency is strong enough **then**

join \( sm \) to \( C \); remove \( sm \) from the set of seed matches

**end**

**while** a seed match is joined to a cluster

Hierarchical clustering

- Several current clusters
- Two clusters are joined in each cycle

**Algorithm** (one example)

make a set of clusters , each seed match being a cluster

**loop**

compare all pairs of clusters, and find the highest scoring pair

**if** the found score is higher than the highest scoring current cluster **then**

remove the two clusters from the cluster set; join the two clusters

**end**

**until** no more clusters are joined
Clustering by use of transformation

Transformation for minimum RMSD usually used

\[ T_1 \text{ define best superposition for } (A_1, B_1) \]
\[ T_2 \text{ define best superposition for } (A_2, B_2) \]
\[ T_1 = T_2 \Rightarrow T_1 \text{ is best transformation for } (A_1 \cup A_2, B_1 \cup B_2) \]

Seed matches

Geometric Hashing f.e. used

\[ A_1 \text{ compatible with } B_1 \text{ (seed match)} \]
\[ A_2 \text{ compatible with } B_2 \text{ (seed match)} \]
The transformation

- The transformation (for best superposition) is defined by 6 parameters: 3 for translation and 3 for rotation

- These are represented as a matrix

- Ideally local clustering criteria can be used, but due to inaccuracy, global is used

- This means that the transformation for the best superposition of \( (A_1 \cup A_2, B_1 \cup B_2) \) is calculated for later cycles

Clustering by use of relation

- The elements are mainly SSEs, represented as sticks

- A seed match is usually a pair of sticks

- To decide if two compatible pairs \((A_i, B_j)\) and \((A_k, B_l)\) are consistent, an intern relation \(\rho\) is used

- The pairs are consistent if \(\rho(A_i, A_k)\) is approximately equal to \(\rho(B_j, B_l)\)

  \(\rho\) should satisfy the following: \(\rho(A_i, A_k)\) and \(\rho(B_j, B_l)\) should be approximately equal if and only if the substructures \((A_i, A_k)\) and \((B_j, B_l)\) are considered as similar

  \(\rho\) should be invariant with respect to rotation and translation, and in most cases to sequential order
Clustering by use of relation

Local clustering criteria is used
Transitivity is generally not fulfilled

Example
Cluster $C = \{(A_rB_j) (A_kB_l)\}$

$(A_rB_j)$ consistent with $(A_sB_i)$
$(A_rB_j)$ not consistent with $(A_kB_l)$

This is the same as $\tilde{n}(A_r, A_i) = \tilde{n}(B_j, B_s)$ but not $\tilde{n}(A_k, A_r) = \tilde{n}(B_l, B_s)$

The relation

- The different methods use a different number of parameters for defining $\rho$

- The number depends on how stringent the constraints for being consistent is defined, i.e. given two sticks, how many parameters are needed for describing the "structural relation" between them? (means changing the value of one of the parameters might change the (sub)structure)

- Checking for consistency using transformation use 6 parameters, hence 6 should be enough. Usually one angle and 2-4 distances are used
Alexandrov and Fisher

Four distances and one angle

\[ \hat{a}_{ik} \text{ is found by projection of } A_i \text{ and } A_k \text{ into a plane parallel to both } A_i \text{ and } A_k \]

---

Alexandrov and Fischer

Requirement for consistency \((A_i, B_j) (A_k, B_l)\)

\[
\begin{align*}
& d_{i}^{\text{min}} - d_{i}^{\text{max}} < \hat{\alpha} \text{ (f.e. } 1.5\text{ Å}) \\
& d_{j}^{\text{min}} - d_{j}^{\text{max}} < \hat{\alpha} \\
& d_{k}^{\text{min}} - d_{k}^{\text{max}} < \hat{\alpha} \\
& d_{l}^{\text{min}} - d_{l}^{\text{max}} < \hat{\alpha} \\
& |\hat{a}_{ik} - \hat{a}_{jl}| < \hat{\alpha}_{\text{lim}}
\end{align*}
\]
Clustering as graph problem

• Make a graph of a structure:
  – The nodes being the elements
  – Node labels being the properties
  – The edges are labeled by the relations
  – Edges between nodes where the relation satisfies some constraints

• Find subgraphs with maximum ”similarity” (isomorphy)
  – Corresponding nodes must be compatible
  – Corresponding edges must be sufficient similar

Node product graph

• If the scoring of the similarity of two substructures is number of compatible pairs, finding the maximum similar substructures is the same as finding a maximal clique in a product graph

• To construct a node product graph
  – Make a node for each compatible pair
  – Make edges between consistent nodes (compatible pairs)
**Node product graph - Example**

- Assume we have the following compatible elements:
  
  \[ \begin{align*}
  a &: (A_1, B_1) \\
  b &: (A_1, B_3) \\
  c &: (A_2, B_2) \\
  d &: (A_2, B_4) \\
  e &: (A_3, B_3) \\
  f &: (A_3, B_1) \\
  g &: (A_4, B_2) \\
  h &: (A_4, B_4)
  \end{align*} \]

- And the following relations are similar:
  \[ \rho(A_1, A_2) = \overline{u}(B_1, B_2) \Rightarrow (A_1, B_1) \text{cons}(A_2, B_2) : a - c \]

- The same for \(a-e, b-f, c-e, d-g\)

**Refinement**

- A final adjustment or refinement may be worth while

- Mainly done on residue level

- An iteration of alternating superposition/alignment is often done
Part IV
Multiple structure alignment and Motif discovery

• Extending pairwise approaches
  – Progressive alignment
  – Alternative methods

• Motif discovery
  – framework
  – the SPratt algorithm

Multiple Alignment

• $N$ sequences of length $m$
• Dynamic programming ($N$-dim matrix): $O(m^N)$
• Becomes unfeasable for, say, $N>5$
Multiple Structure Comparison

- Computational complexity of most pairwise methods does not allow extension to multiple structures.
  - For example, DP based methods.

- Progressive alignment

Example: Extending DDP to multiple alignment

- Align every structure to every other structure - get a similarity matrix - measure of similarity for every pair
- Align the most similar pair - produce consensus structure
- Align the new consensus structure to all others - update similarity matrix
- Iterate until only one consensus structure remains

- Taylor et al 1994
MSSAP: Representing the consensus structures

- Equivalent to a profile in multiple sequence alignment
- In SSAP interatomic vectors are used
- Here: bundles of interatomic vectors - they are represented by average vector and its variance (error measure)

Alternative approach - using a median structure

- Find median structure (the one closest to the all the others)
- Align each of the other structures onto the median by consistently combining the alignments

- Gerstein and Levitt (1998)
Motif Discovery

Unaligned Sequences/structures

Align

Analyze alignment

Pattern Discovery Method

Motif

Framework

Training Structures

Feature extraction

Structure description

Discovery

Solution space
Fitness Algorithm

Comparison

Local/global Scoring scheme
Algorithm

Pattern

Matcher

Score
ye$h$
SPratt - Pattern Driven Algorithm for the Discovery of Structure Motifs

- Discover motifs consisting of a small number of individual residues
  - close in space
  - preserve sequence order.
- Do not use pairwise comparisons - use simultaneously information from all structures.
- Can find motifs recurring in one structure or in a set of structures.


---

SPratt - Idea

- Describe (spatial) local neighbourhoods as strings (Karlin & Zhu, 1997)
- Use the sequence pattern discovery method Pratt to find pattern common to the strings
- Check if string similarity reflects structural similarity
SPratt - Neighbour Strings

- For each residue \( a \) in each structure:
  - \( N_\alpha \): sequence of residues with spatial distance less than \( D \) from \( a \), in order from N-to-C terminal, starting with \( a \).
  - \( C_\alpha \): sequence of residues with spatial distance less than \( D \) from \( a \), in order from C-to-N terminal, starting with \( a \).

Neighbour string for each residue

- Neighbour strings for G:
  - C-string: GT
  - N-string: GYLIWCA

*Remember the index of each residue*
SPratt - Discovery Algorithm

- Encode the neighbourhood of each residue in each structure as 2 strings (N-string and C-string)

- Use Pratt to find patterns in the N strings and the C strings separately

- For each pattern, check if the neighbour string matches reflect structural similarity

SPratt - results

- Can be used to analyse up to around 50 structures to find patterns matching (almost) all.
- Time consuming step: Pratt
Example output: Cystein proteases

<table>
<thead>
<tr>
<th>Protein</th>
<th>1hucAB</th>
<th>1gcb</th>
<th>1fieA</th>
<th>1aim</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>252</td>
<td>452</td>
<td>705</td>
<td>215</td>
</tr>
<tr>
<td>Pattern</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>S220</td>
<td>S393</td>
<td>S397</td>
<td>S176</td>
</tr>
<tr>
<td>[DN]</td>
<td>N219</td>
<td>N392</td>
<td>D396</td>
<td>N175</td>
</tr>
<tr>
<td>H</td>
<td>H199</td>
<td>H369</td>
<td>H373</td>
<td>H159</td>
</tr>
<tr>
<td>F</td>
<td>F32</td>
<td>F76</td>
<td>F317</td>
<td>F28</td>
</tr>
<tr>
<td>C</td>
<td>C29</td>
<td>C73</td>
<td>C314</td>
<td>C25</td>
</tr>
</tbody>
</table>

RMSd matrix

<table>
<thead>
<tr>
<th></th>
<th>1hucAB</th>
<th>1gcb</th>
<th>1fieA</th>
<th>1aim</th>
</tr>
</thead>
<tbody>
<tr>
<td>1hucAB</td>
<td>0.2</td>
<td>1.6</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>1gcb</td>
<td>1.5</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1fieA</td>
<td>1.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1aim</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SPratt: Structures $\rightarrow$ Motif

Combining SPratt with SAP

Pairwise alignment guided by motif

(Taylor & Orengo, 1989; Taylor 1999)
SAP output - cystein proteases

Red: cathepsin-B (1huc),
Green: Human coagulation factor XIII (1fieA)
Cold (blue): bad superposition
Hot (red, yellow): good fit
SAP output - 2Fe2S Ferredoxins

- Red: (1alo), Green: (2pia), Cold (blue): bad superposition, Hot (red, yellow): good fit

SPratt2

- Same ideas as SPratt in that spatial neighbourhoods are encoded as neighbour strings.
- Instead of using Pratt, a new built-in search procedure is used.
  - Use structural information in search
  - Different (string) pattern solution space

- More efficient
  - can take more structures as input
  - can do more general pattern classes

Neighbour strings in SPratt2

- One neighbour string per residue including all residues within $D$ Ångström in N-to-C direction.

Neighbour string for G:

ACWILYGT

Neighbour String Angle Constraint

- C-alpha

$\cos(\lambda_i) = \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|}$

Include residue $j$ in $i$’s neighbour string only if $\lambda_i > 0$ and $\lambda_j > 0$ (or another threshold)
Neighbour String *-Patterns

- String patterns consist of
  - single amino acids
  - match sets
  - * which matches an arbitrary number of consecutive amino acids

- Example:
  - S*[DN]*H*F*C

- Different from SPratt that uses Prosite style patterns, e.g.
  - S-x(2,3)-[DN]-x(3,5)-H-x-F-x(8,10)-C

Search Problem in SPratt2

- Find *-patterns common to neighbour strings of at least $k$ structures
- Each patterns contains one “center residue” - matching neighbour strings have their center residue aligned with that of the pattern.
- The structural conformation of the residues in the occurrences of a pattern should be similar.
Probe → *-Pattern

Search problem:
Find *-pattern generalisation of the probe so that
- it matches neighbour strings from at least \( k \) structures
- the structures of the matches can be superposed onto the probe
  structure within \( d \) angstroms

*-Pattern Exploration
Seeded by Probe

Search tree:

\[ \begin{align*}
& \text{ACWILYGT} \\
& \quad \text{G} \\
& \quad \quad \text{G*T} \quad \text{Y*G} \quad \text{L*G} \quad \text{I*G} \\
& \quad \quad \quad \text{Y*G*T} \quad \text{L*Y*G} \quad \text{I*L*G} \quad \text{W*I*G} \\
& \quad \quad \quad \quad \text{L*Y*G*T} \quad \text{C*W*I*G}
\end{align*} \]
Use of Probe Structure in Search

- Matches that cannot be superposed onto the probe within $d$ angstrom, are discarded - not counted.
- Internal distance RMSd is used since this is easy to compute.
- Used only when *-patterns have 4 or more components (limit to be explored with respect to speed and sensitivity).

Which Probes?

- If $N$ structures are analysed and patterns matching at least $k$ structures are sought, use as probes all neighbour strings from the smallest $n-k+1$ structures.

- If a query structure is given, generate probes only from this.
Ranking of patterns

- The neighbour string pattern has an information content $I$.
- The matches can be superposed onto each other giving a matrix of RMSd values.

$$Score = \frac{I}{\text{max RMSD}}$$

- Increases with increasing number of residues in pattern
- Decreases with softening amino acid constraints
- Decreases with poorer structural fit

Implementation

- *One* program - string pattern discovery has access to structure information
- Minimize memory usage
  - output patterns as they are found - do post-processing externally
- Allows mining of non-redundant subset of PDB
  - e.g., more than 2000 chains simultaneously
Performance of SPratt2 compared to that of SPratt

<table>
<thead>
<tr>
<th>Family</th>
<th>k/N</th>
<th>av. len</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cupredoxins</td>
<td>10/10</td>
<td>263</td>
<td>2 min → 6 sec</td>
</tr>
<tr>
<td>Cys. prot.</td>
<td>5/5</td>
<td>560</td>
<td>58 min → 16 sec</td>
</tr>
<tr>
<td>4Fe-4S</td>
<td>5/5</td>
<td>86</td>
<td>11 sec → 1 sec</td>
</tr>
<tr>
<td>Ser. Prot</td>
<td>10/12</td>
<td>231</td>
<td>12 min → 57 sec</td>
</tr>
</tbody>
</table>

Mining PDB

- Maximum 30% identity non-redundant subset of PDB, resolution 2.0 or better: 779 chains (culledPDB, 18/5/00)
- Radius 10 Ångstrøm
- Min-angle: λ>0
- Max. RMSd: 1 Å
- Memory usage: < 200 Mbyte
- k: minimum nr matching structures
Mining PDB - time usage and number of patterns found

Figure 1: A: Running time of SFrank2 (in seconds — vertical axis) is shown when applied to PDB (see text) with different values of $k$ (minimum support requirement — horizontal axis). We see that for $k = 2$ SFrank2 takes almost 8 hours while for $k = 20$, the run takes around 4 hours. B: The number of patterns (vertical axis) produced by SFrank2 for different values of $k$ (horizontal axis).

Need to Reduce Pattern Set:
Select a “Covering” Subset of Patterns

- Discard patterns with score < 20
- Pick pattern with maximum
  - Score
  - Support (number of matching structures)
- Remove patterns matching the same structures.
- Pick maximum pattern in remaining pattern set.
- Keep going until no more patterns to be picked.
- “Representative subset” of the patterns - remove patterns matching the same structure set
Conclusions

• Protein 3D structures are more complex than sequences
• Protein structure comparison and alignment is complex
• We have
  – presented a framework
  – reviewed some methods
• A lot of methods out there

• Have not touched too much on
  – scoring, assessment

Conclusions (cont’d)

• Motif discovery useful alternative approach to alignment
  – can identify patterns not significant when found between two structures
  – avoids laborious and in-direct pairwise comparisons

• Motif discovery by (unsupervised) data mining feasible

• Benchmarks could be useful, but difficult to find set of common interest
  – Protein family databases (esp. SCOP) can be used for assessment
Open Problems and Future Directions

- Will soon have all (most) protein structures (experimentally or by homology modelling).
- Gives increased importance to structure comparison, classification, and motif discovery.

- Need more powerful methods for
  - analysing large data sets
  - finding subtle (remote) similarities

Acknowledgments

- Willie Taylor (NIMR London and UoB)
- Darrell Conklin (Zymogenetics, Seattle, USA)
- David Gilbert (City University, London, UK)
- Alvis Brazma (EMBL-EBI, Cambridge, UK)

http://www.ii.uib.no/forskningsgrupper/bio