Week 5
Protein Structure I: Pairwise Alignment

CS 426
Fall 2003

Sequence (as a String) vs. Structure

A protein is a sequence: a string over an alphabet of 20 characters (the 20 amino acids)

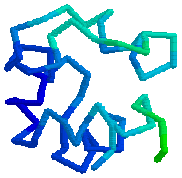
- View of a protein as a string leads to powerful tools
  - String matching using Dynamic Programming
  - Approximate string matching
  - Suffix trees
  - ...

A protein is a structure: a 3D geometric shape

- Our goal is to build similar tools to manipulate/analyze shapes as easily as we do strings

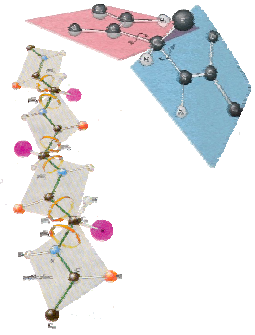
How Do We Measure Protein Shape?

- Protein function is largely based on the proteins' geometric shape
  - Protein substructures with similar shapes are likely to share a common function (e.g., a binding site)
- How do we measure protein shape?
  - Compare pictures?
  - Bond angles?
  - Distance matrices?
  - RMSD (Root Mean Square Distance)?
  - Geometric Hashing?
  - ?



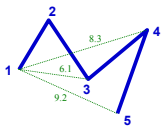
A Local Shape Representation: Torsion Angles

- Torsion angles: (φ, ψ)
  - An example of a purely local representation
  - Invariant with respect to rotation or translation
  - Compact: size O(n) for protein with n residues
- But local similarity does not always imply global similarity
  - E.g., minor differences accumulate



Distance Matrices

- Matrix of pairwise distances between selected atoms
- Invariant with respect to rotation or translation
- Compare proteins by comparing their distance matrices

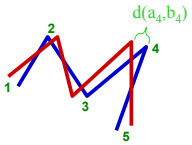


- Disadvantages:
  - A distance matrix has size $\Theta(n^2)$ for a protein with n residues
  - Comparing distance matrices is hard (NP-Complete: Subgraph Isomorphism)

	1	2	3	4	5
1	0.0	3.8	6.1	8.3	9.2
2	3.8	0.0	3.8	7.7	8.0
3	6.1	3.8	0.0	3.8	5.2
4	8.3	7.7	3.8	0.0	3.8
5	9.2	8.0	5.2	3.8	0.0

Root Mean Square Distance (RMSD)

- Method for comparing similarity of two point sets (e.g., atoms or α-carbons)
  - Call the sets A and B
- Need a 1-to-1 correspondence between points of each set
  - For a_i in set A, the corresponding point in set B is b_i
- $D_{RMS}(A,B) = ((1/n)\Sigma_i d^2(a_i,b_i))^{1/2}$
  - $d^2(a_i,b_i)$ is the squared distance between a_i and b_i
  - n = |A| = |B|



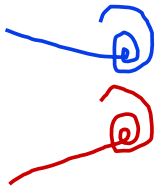
- Really want the *minimum* RMSD
  - In other words, we want to translate and rotate set B to bring it as close as possible to set A
  - This seems like it should be hard to do

RMSD Algorithm

- Input
  - Proteins A and B given by the 3D coordinates of selected atoms (e.g., the α-carbons along the protein backbone)
  - 1-to-1 correspondence between the atoms
- Algorithm [O(n) time]
  - Align "centers-of-mass"
    - In other words, we average the points of A and we average the points of B, then we translate B to make those averages the same
    - Some simple calculus shows this is optimal
  - Determine best possible rotation
    - Involves computing the SVD (Singular Value Decomposition) of a 3x3 matrix, the *correlation matrix*

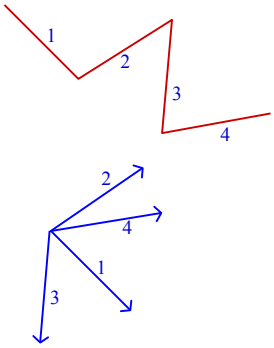
RMSD Disadvantages

- Sensitive to outliers
  - A few "bad" matches can swamp the good matches
- Portions of the protein are weighted unequally
  - Portions far from the center of mass are, in effect, weighted more heavily
- Thus, RMSD works best when comparing proteins that are known to be similar



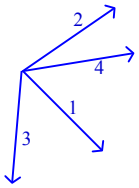
Protein Shape with URMS Distance

- Transform α-carbon backbones into sequences of unit vectors
  - Distance between consecutive α-carbons is constant (about 3.8 Å)
- Compute the minimum RMS distance for the two sequences of unit vectors
  - Use rotation only; no translation
- We call this the *U*nit-vector *RMS* (URMS) Distance



URMS Advantages

- Insensitive to outliers
  - No mismatch can have distance greater than 2
- Weighs all portions of the protein equally
  - All the unit vectors share the same origin
- Computationally efficient
  - O(n) (like RMSD)
  - Can also do groups of comparisons efficiently



Another URMS Advantage

- Has an upper bound (independent of n)
  - Two unit-vectors can be no farther apart than 2 units; thus, maximum possible URMS distance is 2
  - The standard RMS distance can be arbitrarily large
- Does not grow significantly with the length of the protein
  - For the standard RMS, larger proteins have larger RMS distances; thus, there is no standard value implying "close"
  - Expected URMS for random pseudo-proteins is $(2–2.84/n^{½})^{½}$

Detecting Common Substructures

- Goal: Detect substructures within proteins that have the same or similar shape
  - Similar shape indicates similar function
  - Substructures can have similar shape even when sequence resemblance is small (< 20%)
- Intuition: proteins that are unrelated should "look random" to each other

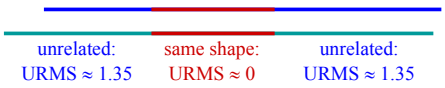
Random Proteins and the URMS

- Given two random (pseudo-)proteins, what is their expected URMS distance?
  - Partial derivation shows answer is $(2–2.84/n^{1/4})^{1/2}$
  - Derivation is incomplete due to need for (false) assumption of independence
- Experimental results (on random pseudo-proteins)

n	$URMS_f$	$URMS_e$
100	1.310	1.311
200	1.341	1.341
300	1.355	1.354
400	1.363	1.363
500	1.369	1.368

Finding Matching Substructures

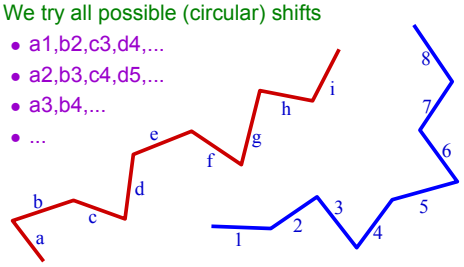
- Consider two proteins with contiguous portions that have the same shape
- Intuition: unrelated protein portions should "look random" to each other



- If enough (about 10-15%) of the proteins are of similar shape then we should see a significant drop in the URMS distance at the correct shift

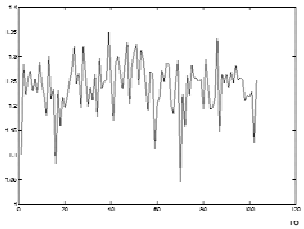
Finding the Right Shift

- A *shift* determines the correspondence between the two proteins
- We try all possible (circular) shifts
  - a1,b2,c3,d4,...
  - a2,b3,c4,d5,...
  - a3,b4,...
  - ...

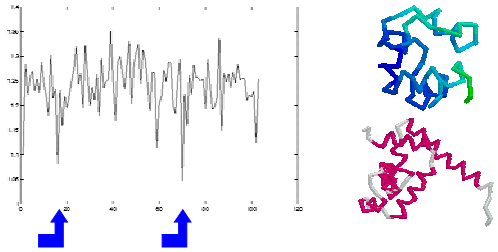


Experiments on Real Proteins

- Experiment comparing 2cro (65 residues) with 2wrp (104 residues)
- Graph shows URMS value for 2cro compared against each length-65 substring of 2wrp
- Prediction for random proteins: 1.284
- Result here is closer to 1.25
- Conclusion: real proteins aren't random (but we knew this)



URMS as a Function of Shift



- Likely substructure matches at shifts 17 and 70 (2cro vs. 2wrp)

We know the right shift, but...

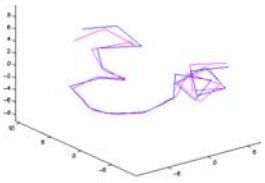
- It's not enough to just know the shift
- Intuition: If a chunk of protein A is shaped like a chunk of protein B then a single 3D rotation will bring these chunks into alignment
- Idea:
  - Determine rotation matrix needed to map an adjacent pair of unit vectors in A to corresponding pair in B
  - Look for sequences of such pairs that have similar rotation matrices

Comparing Rotations

- We use the Frobenius norm to compare rotation matrices
  - Easy to compute
  - Independent of reference frame
- Theoretically, it would be better to find cliques
  - Too time-consuming (NP-hard)
  - Experiments show consecutive comparisons work very well

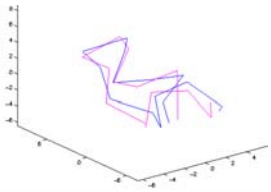
2cro-2wrp Matching Substructures

Shift 17



2cro (17-40) and 2wrp (62-85)

Shift 70



2cro (48-65) and 2wrp (65-82)

Substructure Matching Algorithm

To find substructures common to proteins A and B:

- Determine the URMS distance for each shift of A against B
- Choose the k shifts with the lowest URMS distances
- For each of these k shifts, determine if there is a contiguous chunk with similar rotation matrices
- If such a contiguous chunk exists then we have found a match

Algorithm Observations

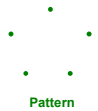
- The FFT can be used to find the URMS distance for each possible shift in time O(n log n)
- Experiment indicates that k larger than about 10-20 is useless
- Usually, a low URMS distance indicates a large matching substructure
  - But sometimes a low URMS distance is caused by several small matching substructures

Finding Geometric Patterns using Hashing

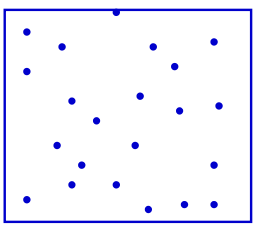
- Goal: determine if a particular pattern of points appears in a scene of many points
  - Finding a particular pattern of stars in a sky-photo
  - Finding a tumor in a catscan
  - Finding a pattern of atoms (an active site) on the surface of a protein
- Basic idea:
  - Try every possible way to place the pattern on the scene
  - The position that has the most hits is the best match
- Hit ≡ pattern point is "close enough" to a scene point

A Pattern Matching Example

- Does the *pattern* appear in the *scene*?
- Assume translation, but no rotation



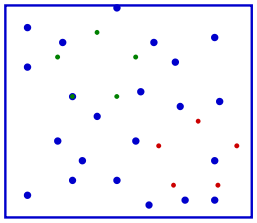
Pattern



Scene

Finding Geometric Patterns Efficiently

- Don't look at all pattern positions
- Just look at the ones with at least one hit
- For this example (translation only), we try each pattern point on each scene point
  - There are *mn* possibilities where *m* is the size of the pattern and *n* is the size of the scene



A Simple Pattern Matching Algorithm

```
                for each pattern point p
                  for each scene point s
O(mn)               Determine translation so that p falls on s;
O(mn)               count = 0;
                    for each translated pattern point q
O(m²n²)               Find the scene point r closest to q;
O(m²n)                if (r is "close enough" to q) count++;
O(mn)               Remember which p,s gives max count;
```

Total time = $O(m^2n^2)$

How Do We Detect a *Hit*?

- Can't afford to check every point in the scene to see if it's close enough to a pattern point
- One solution is to *rasterize* the scene
  - Place a grid on the scene
  - Each grid-square is a bin
  - Points are "close enough" if they are in the same bin
- Disadvantage:
  - We've lost accuracy
  - We may miss some nearby points
- Advantage:
  - Instead of searching for arbitrary points, we search for occupied bins
  - The bins follow a regular pattern
  - The bins can be stored as an array !

Why Check Every Pattern Point?

- We could save time (a factor of m) by choosing just one pattern point p to translate to each scene point s, but…
  - Some pattern points may be missing from the scene
    - Due to occlusion or errors in the scene data, for example
  - The pattern may not align well with the bins

An Improved Pattern Matching Algorithm

```
O(b)    Clear all the bins;
        for each scene point s
O(n)      Mark the bin for s as occupied;
        for each pattern point p
          for each scene point s
O(mn)       Determine translation so that p falls on s;
O(mn)       count = 0;
            for each translated pattern point q
O(m²n)        if (q falls in an occupied bin) count++;
O(mn)       Remember which p,s gives max count;
```

Total time = $O(b+m^2n)$

Is the Improved Algorithm Better?

- Total time is now $O(b+m^2n)$ instead of $O(m^2n^2)$
  - b is the number of bins
  - Note: m is small; n is *big*
- But we have lost accuracy and the array of bins can be expensive
  - Can be a very large array
  - Most bins are empty anyway
  - Note: could use a bit-array, but it's still large

Improvement Using Hashing

Idea:
- Don't keep an array of bins
- Instead, place the *occupied* bins in a hash table
- Hash on the bin's array indices

Result:
- Initialization takes time O(n) instead of O(b)
- Still fast to test whether q falls in an occupied bin
  - A single test takes *expected* time O(1) instead of *worst-case* time O(1)
- Algorithm now runs in *expected* time $O(m^2n)$ instead of *worst-case* time $O(b+m^2n)$

Both Translation and Rotation?

- A single pattern point p and single scene point s aren't enough to determine a transformation
- Instead, we need 2 pattern points and 2 scene point
  - The first (p,s) pair gives the translation; the second gives the rotation
  - This leads to an algorithm that runs in expected time $O(m^3n^2)$
- If more-complicated transformations are allowed then more points are needed and the time bound grows
  - Example: For 3D translation and rotation, three points are needed and the time bound is $O(m^4n^3)$

Geometric Hashing

- Problem: A running time of $O(m^3n^2)$ (this is the time bound assuming both translation and rotation in 2D) may be impractical
- Idea: Use hashing to separate the runtime factors

- Preprocessing: [$O(m^3)$ time]

```
for each pair (p,q) of pattern points
  Transform all pattern points to reference frame (p,q)
    [i.e., reference frame with origin at p, +X-axis through q];
  for each pattern point r
    Determine bin containing the transformed r
    Place (p,q) in Hashtable, hashed by bin number of r
```


Geometric Hashing (continued)

- To test for a match: [$O(n^3)$ time]

```
for each pair (s,t) of scene points
  Find transformation with origin at s, +X-axis through t;
  for each transformed scene point u
  if u falls in an occupied pattern bin [tested using hashing]
    Increment the vote for that pattern transformation;
  Report a match if some transformation gets enough votes;
```


Geometric Hashing Advantages

- Advantages
  - Depending on problem, $O(n^3)$ may be significantly better than $O(m^3n^2)$
  - If we want to search for multiple patterns, this affects the preprocessing time, but not the matching time
- Time is better in practice than big-O bound
  - Can often reject a pair (s,t) of scene points quickly because, for instance, they're too far apart to match any pair of pattern points

Applications of Geometric Hashing

- Applications to
  - Computer vision
  - Astronomy
  - Medical imaging
  - Fingerprint matching
  - Molecular docking
  - Protein structure matching
- References:
  - Lamdan & Wolfson, Geometric hashing: A general and efficient model-based recognition scheme (1988); lots of papers since then
  - Web tutorial on protein structure comparison
    - http://www.ii.uib.no/~inge/talks/ismb-tutorial/
    - Includes geometric hashing (among other methods)

Protein Structure Comparison Summary

- Methods discussed (a subset of the many available)
  - Compare pictures
  - Torsion Angles
  - Distance Matrices
  - RMSD (Root Mean Square Distance)
    - We talked about using it on α-carbons
    - Has also been used to compare Torsion Angles and Distance Matrices
  - URMS (Unit-vector Root Mean Square)
  - Geometric Hashing
- Why so many methods?
  - Proteins are complex three-dimensional structures
  - Choice depends on
    - Purpose of comparison
    - Whether method is appropriate
- Protein backbone only
  - Torsion Angles, URMS
- Any cloud of points (or atoms)
  - Distance Matrices, RMSD, Geometric Hashing

URMS is Being Used

- *CASP* (*C*ritical *A*ssessment of Techniques for Protein *S*tructure *P*rediction)
  - A "contest" for researchers trying to predict protein structure from sequence
  - Held every 2 years
  - CASP5 was held in 2002
  - Entrants are asked to predict protein structures that are known, but not yet published
  - Evaluation is done by "experts"
- *CAFASP* (*C*ritical *A*ssessment of *F*ully *A*utomated *S*tructure *P*rediction)
  - Held in conjunction with CASP
  - Only automated web servers are eligible
  - CAFASP3 was held in 2002
  - Evaluation is automated
    - URMS has been used as part of evaluation process for both CAFASP2 and CAFASP3