CSE 589
Applied Algorithms
Spring 1999

Approximate String Matching
Contiguous Ordering - PQ Trees

DNA

- DNA is a large molecule that can be abstractly defined as a sequence of symbols from the set, A, C, G, T, called nucleotides.
- The human genome has about 3 billion nucleotides.
  - A huge percentage of the genome is shared by all humans.
  - Some of the variation makes us different.
  - Some of the variation is inconsequential.
  - The human genome is still being discovered.

Approximate Matching

- Two DNA sequences approximately match if one can be transformed into the other by a short sequence of replacements and insertions of gaps.
- Example:
  - `S = AGCATG`
  - `T = AGATCGT`
- Approximate matching - is a gap
  - `S′ = A G - - C A T G`
  - `T′ = A G A T C G T -`

Applications of Approximate Matching

- DNA string alignment.
  - Given two similar DNA sequences find the best way to align them to the same length.
- DNA database searching.
  - Find DNA sequences that are similar to the query.
- Approximate text matching for searching.
  - agrep in unix
- Spell checking
  - Find the words that most closely match the misspelled word.

Scoring an Approximate Matching

- We need a way of scoring the quality of an approximate matching.
- A scoring function is a mapping σ from $\{A, C, G, T, -\}^2$ to integers.
  - The quantity σ(x,y) is the score of a pair of symbols, x and y.
- Example:
  - σ(x,y) = +2 if x=y and x in {A,C,G,T}
  - σ(x,y) = -1 otherwise

Scoring Example

- Example:
  - `S′ = A G - - C A T G`
  - `T′ = A G A T C G T -`
- Score = 4 x 2 + 4 x(-1) = 4
- Is this the best match between the two strings with this scoring function?
  - `S = AGCATG`
  - `T = AGATCGT`

Approximate String Matching Problem

- Input: Two strings S and T in an alphabet Σ and a scoring function σ.
- Output: Two strings S' and T' in the alphabet Σ' = Σ union {-} with the properties:
  - S = S' with the -'s removed.
  - T = T' with the -'s removed.
  - |S'| = |T'|
  - The score $\sum_{i=1}^{|S'|} \sigma(S'[i], T'[i])$ is maximized.

Algorithms for Approximate String Matching

- O(mn) time and storage algorithm (using dynamic programming) invented by Needleman and Wunch, 1970.
- Fischer and Paterson, 1974, invented a very similar algorithm for computing the minimum edit distance between two strings.

Dynamic Programming for Approximate String Matching

- Assume S has length m and T has length n.
- For all i and j, 0 ≤ i ≤ m and 0 ≤ j ≤ n, we find the maximum score for the sequences S[1..i] and T[1..j].
- The "dynamic program" fills in a (m+1)x(n+1) matrix M in increasing order of i and j with these maximum values.
- Once the dynamic program has completed we can recover the optimal string S' and T' from the matrix M.

Max Score Recurrence

- Define M[i,j] = maximum score for a match between S[1..i] and T[1..j].

$$M[i,0] = \sum_{k=1}^{i} \sigma(S[k], -)$$ match of S[1..i] with empty string

$$M[0,j] = \sum_{k=1}^{j} \sigma(-, T[k])$$ match of T[1..j] with empty string

$$M[i,j] = \max\{$$
$$M[i-1, j-1] + \sigma(S[i], T[j]),$$
$$M[i-1, j] + \sigma(S[i], -),$$
$$M[i, j-1] + \sigma(-, T[j])\}$$

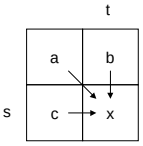
Dynamic Program Initialization

```
S = AGCATG
T = AGATCGT
```

scoring function
+2 for exact match
-1 otherwise

		0	1 A	2 G	3 A	4 T	5 C	6 G	7 T
0		0	-1	-2	-3	-4	-5	-6	-7
1	A	-1							
2	G	-2							
3	C	-3							
4	A	-4							
5	T	-5							
6	G	-6							

The Dynamic Programming Pattern



d = a + 2 if s = t
= a - 1 otherwise

h = c - 1

v = b - 1

x = max(d, h, v)

Dynamic Program Example (1)

S = AGCATG
T = AGATCGT

scoring function
+2 for exact match
-1 otherwise

		0	A	1 G	2 A	3 T	4 C	5 G	6 T	7
0		0	-1	-2	-3	-4	-5	-6	-7
1	A	-1	2						
2	G	-2							
3	C	-3							
4	A	-4							
5	T	-5							
6	G	-6							

Dynamic Program Example (2)

S = AGCATG
T = AGATCGT

scoring function
+2 for exact match
-1 otherwise

		0	A	1 G	2 A	3 T	4 C	5 G	6 T	7
0		0	-1	-2	-3	-4	-5	-6	-7
1	A	-1	2	1					
2	G	-2	1						
3	C	-3							
4	A	-4							
5	T	-5							
6	G	-6							

Dynamic Program Example (3)

S = AGCATG
T = AGATCGT

scoring function
+2 for exact match
-1 otherwise

		0	A	1 G	2 A	3 T	4 C	5 G	6 T	7
0		0	-1	-2	-3	-4	-5	-6	-7
1	A	-1	2	1	0				
2	G	-2	1	4					
3	C	-3	0						
4	A	-4							
5	T	-5							
6	G	-6							

Dynamic Program Example (4)

S = AGCATG
T = AGATCGT

scoring function
+2 for exact match
-1 otherwise

		0	A	1 G	2 A	3 T	4 C	5 G	6 T	7
0		0	-1	-2	-3	-4	-5	-6	-7	
1	A	-1	2	1	0	-1	-2	-3	-4	
2	G	-2	1	4	3	2	1	0	-1	
3	C	-3	0	3	3	2	4	3	2	
4	A	-4	-1	2	5	4	3	3	2	
5	T	-5	-2	1	4	7	6	5		
6	G	-6	-3	0	3	6	6			

Dynamic Program Example (5)

S = AGCATG
T = AGATCGT

scoring function
+2 for exact match
-1 otherwise

		0	A	1 G	2 A	3 T	4 C	5 G	6 T	7
0		0	-1	-2	-3	-4	-5	-6	-7	
1	A	-1	2	1	0	-1	-2	-3	-4	
2	G	-2	1	4	3	2	1	0	-1	
3	C	-3	0	3	3	2	4	3	2	
4	A	-4	-1	2	5	4	3	3	2	
5	T	-5	-2	1	4	7	6	5	5	
6	G	-6	-3	0	3	6	6	8	7	

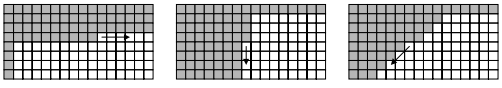
Max score for any matching

Dynamic Programming Order

By row
for i = 1 to m do
  for j = 1 to n do
    M[i,j] := ...

By column
for j = 1 to n do
  for i = 1 to m do
    M[i,j] := ...

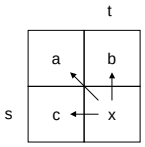
By diagonal



Which order is best?

How to Find the Matching

- To find S' and T' we build a matching graph.



x = a + 2 if s = t
= a - 1 otherwise ?

x = c - 1 ?

x = b - 1 ?

If the answer is yes,
include the corresponding edge.

Computing the Matching Graph (1)

		0	1 A	2 G	3 A	4 T	5 C	6 G	7 T
0		0	-1	-2	-3	-4	-5	-6	-7
1	A	-1	2	1	0	-1	-2	-3	-4
2	G	-2	1	4	3	2	1	0	-1
3	C	-3	0	3	3	2	4	3	2
4	A	-4	-1	2	5	4	3	3	2
5	T	-5	-2	1	4	7	6	5	5
6	G	-6	-3	0	3	6	6	8 ←	7

Computing the Matching Graph (2)

		0	1 A	2 G	3 A	4 T	5 C	6 G	7 T
0		0	-1	-2	-3	-4	-5	-6	-7
1	A	-1	2	1	0	-1	-2	-3	-4
2	G	-2	1	4	3	2	1	0	-1
3	C	-3	0	3	3	2	4	3	2
4	A	-4	-1	2	5	4	3	3	2
5	T	-5	-2	1	4	7	6	5	5
6	G	-6	-3	0	3	6	6	8 ←	7

Computing the Matching Graph (3)

		0	1 A	2 G	3 A	4 T	5 C	6 G	7 T
0		0	-1	-2	-3	-4	-5	-6	-7
1	A	-1	2	1	0	-1	-2	-3	-4
2	G	-2	1	4	3	2	1	0	-1
3	C	-3	0	3	3	2	4	3	2
4	A	-4	-1	2	5	4	3	3	2
5	T	-5	-2	1	4	7 ←	6	5	5
6	G	-6	-3	0	3	6	6	8 ←	7

Computing the Matching Graph

		0	1 A	2 G	3 A	4 T	5 C	6 G	7 T
0		0	-1	-2	-3	-4	-5	-6	-7
1	A	-1	2	1	0	-1	-2	-3	-4
2	G	-2	1	4	3	2	1	0	-1
3	C	-3	0	3	3	2	4	3	2
4	A	-4	-1	2	5	4	3	3	2
5	T	-5	-2	1	4	7 ←	6	5	5
6	G	-6	-3	0	3	6	6	8 ←	7

Computing the Matching Path

		0	1 A	2 G	3 A	4 T	5 C	6 G	7 T
0		0	-1	-2	-3	-4	-5	-6	-7
1	A	-1	2	1	0	-1	-2	-3	-4
2	G	-2	1	4	3	2	1	0	-1
3	C	-3	0	3	3	2	4	3	2
4	A	-4	-1	2	5	4	3	3	2
5	T	-5	-2	1	4	7 ←	6	5	5
6	G	-6	-3	0	3	6	6	8 ←	7

Matching Path
(0,0)
(1,1)
(2,2)
(3,2)
(4,3)
(5,4)
(5,5)
(6,6)
(6,7)

There can be multiple paths

Algorithm to find Matching

- Follow any path in the matching graph starting at (m,n).
- The path will end up at (0,0).
- Output each pair (i,j) visited to make a list of pairs forming a matching path.

Computing the Matching

```
p = length of the matching path P
i :=1;
j := 1;
for k = 1 to p do
   if P[k].first = P[k-1].first then
      S'[k] := - ;
   else
      S'[k] :=  S[i];
       i := i + 1;
   if P[k].second = P[k-1].second then
      T'[k] := - ;
   else
      T'[k] :=  T[j];
       j := j + 1;
```

P	
0	(0,0)
1	(1,1)
2	(2,2)
3	(3,2)
4	(4,3)
5	(5,4)
6	(5,5)
7	(6,6)
8	(6,7)

first second

Creating the Matching

P	
0	(0,0)
1	(1,1)
2	(2,2)
3	(3,2)
4	(4,3)
5	(5,4)
6	(5,5)
7	(6,6)
8	(6,7)

```
    1 2 3 4 5 6
S = A G C A T G
T = A G A T C G T

S'= A G C A T - G -
T'= A G - A T C G T
```

Score = 5 x 2 + 3 x (-1) = 7

Example of Multiple Paths

		0	1	2	3	4	5
			C	A	T	G	T
0		0	-1	-2	-3	-4	-5
1	A	-1	-1	1	0	-1	-2
2	C	-2	1	0	0	-1	-2
3	G	-3	0	0	-1	2	1
4	C	-4	-1	-1	-1	1	1
5	T	-5	-2	-2	1	0	3
6	G	-6	-3	-3	0	3	2

Multiple matching with same score

```
- A C G C T G
C A T G - T -

A C G C T G -
- C A - T G T

A C G C T G -
- - C A T G T
```

score = 3 x 2 + 4 x (-1) = 2

Approximate String Searching

- Input: Query string Q and target string T in an alphabet Σ and a scoring function σ, and a minimum score r.
- Output: The set of k such that for some i ≤ k score(Q,T[i..k]) ≥ r. That is, an approximate match of some substring of T that ends at index k has a score of at least r.
  - score(X,Y) is the maximum score for all matchings between X and Y.

Search Algorithm

- We change the previous dynamic program slightly.

$$M[i,0] = \sum_{k=1}^{i} \sigma(Q[k],-)$$

$M[0,j] = 0$ We don't care where the match begins in T

$$M[i,j] = \max\{$$
$$M[i-1,j-1] + \sigma(Q[i],T[j]),$$
$$M[i-1,j] + \sigma(Q[i],-),$$
$$M[i,j-1] + \sigma(-,T[j])\}$$

Choose all k such that $M[m,k] \geq r$ where m is the length of Q.

Example of Approximate Matching

```
Q = AGTA
T = AGATCGTAGT
```

r = 5

scoring function
+2 for exact match
-1 otherwise

		0	1	2	3	4	5	6	7	8	9	10
			A	G	A	T	C	G	T	A	G	T
0		0	0	0	0	0	0	0	0	0	0	0
1	A	-1	2	1	2	1	0	-1	-1	2	1	0
2	G	-2	1	4	3	2	1	2	1	1	4	3
3	T	-3	0	3	3	5	4	3	4	3	3	6
4	A	-4	-1	2	5	4	4	3	3	6	5	5

output is 3, 8, 9, 10

Recovering the Matchings

```
Q = AGTA
T = AGATCGTAGT
```

		0	1	2	3	4	5	6	7	8	9	10
			A	G	A	T	C	G	T	A	G	T
0		0	0	0	0	0	0	0	0	0	0	0
1	A	-1	2	1	2	1	0	-1	-1	2	1	0
2	G	-2	1	4	3	2	1	2	1	1	4	3
3	T	-3	0	3	3	5	4	3	4	3	3	6
4	A	-4	-1	2	5	4	4	3	3	6	5	5

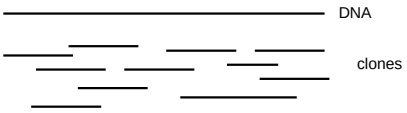
```
Q  AGTA              Q  A--GTA-
T  AG-A 1-3          T  ATCGTAG 3-9

Q  A--GTA            Q  AGTA
T  ATCGTA 3-8        T  AGT- 8-10
```


Notes on Approximate Matching

- Time complexity O(mn)
- Storage complexity O(mn)
  - Storage in the dynamic program can be reduced to O(m+n) by just keeping the frontier.
  - Recovering the matching can be done in time O(m+n) cleverly.

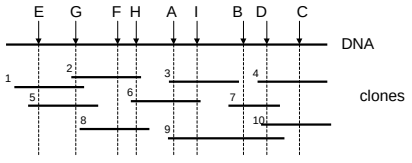
DNA Sequence Reconstruction

- DNA can only be sequenced in relatively small pieces, up to about 1,000 nucleotides.
- By chemistry a much longer DNA sequence can be broken up into overlapping sequences called clones.



Tagging the Clones

- By chemistry the clones can be tagged by identifying a region of the DNA uniquely.



- Each clone is then tagged correspondingly.

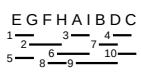
Problem to Solve

- Given a set of tagged clones, find a consistent ordering of the tags that determines a possible ordering of the DNA molecule.

input

1. {E, G}
2. {F, G , H}
3. {A, I}
4. {C, D}
5. {E, G}
6. {A, H, I}
7. {B, D}
8. {F, H}
9. {A, B, D, I}
10. {C, D}

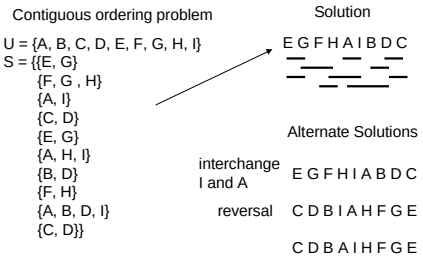
output



Contiguous Ordering Problem

- Input: A universe U and a set S = {S_1, S_2, ..., S_m} of subsets of U.
- Output: An ordering, b_1,b_2, ... , b_n, of U such that for all i, S_i = {b_j, b_{j+1}, ..., b_{j+k}} for some j and k. That is each set in S is contiguous in the ordering of U.
- In this terminology the U is the set of tags and S is the set of tagged clones.

Contiguous Ordering Solutions



Linear Time Algorithm

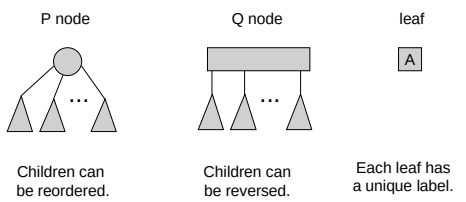
- Booth and Lueker, 1976, designed an algorithm that runs in time O(n+s).
  - n is the size of the universe and s is the sum of the sizes of the sets.
- It requires a novel data structure called the PQ tree that represents a set of orderings.
- PQ trees can also be used to test whether an undirected graph is planar.

DNA Downside

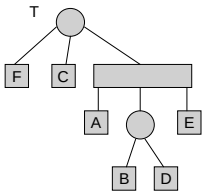
- Tagging can have errors.
  - A tag might be omitted from a clone
  - A clone might be identified as having a tag when it doesn't
  - A tag might be duplicated, not be unique.
- In all these cases finding the "best" ordering of the tags in an NP-hard problem.
  - In our perfect world without errors, ordering can be done in linear time.

PQ Trees

- PQ trees are built from three types of nodes



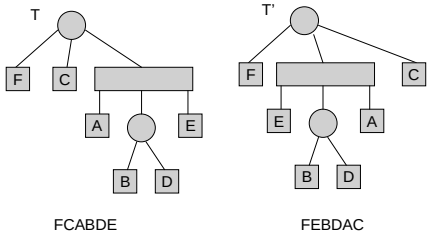
Example PQ-Tree



The frontier of T defines the ordering F(T) = FCABDE, just read the leaves left to right.

T' is equivalent to T if T can be transformed into T by reordering the children of P nodes and reversing the children of Q nodes.

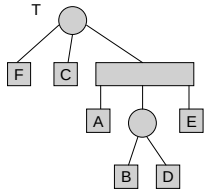
Equivalent PQ Trees



FCABDE FEBDAC

Orderings Defined by a PQ Tree

- Given a PQ tree T the orderings defined by T is
  - PQ(T) ={F(T') : T' is equivalent to T}



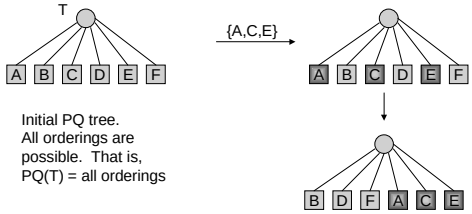
There are 6 x 2 x 2 = 24 distinct orderings in PQ(T).

PQ Tree Solution for the Contiguous Ordering Problem

- Input: A universe U and a set S = {S_1, S_2, ..., S_m} of subsets of U.
- Output: A PQ tree T with leaves U with the property that PQ(T) is the set of all orderings of U where each set in S is contiguous in the ordering.

Example (1)

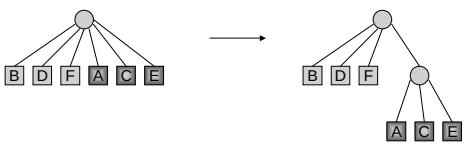
U = {A,B,C,D,E,F}
S = {{A,C,E}, {A,C,F}, {B,D,E}}



Initial PQ tree.
All orderings are possible. That is, PQ(T) = all orderings

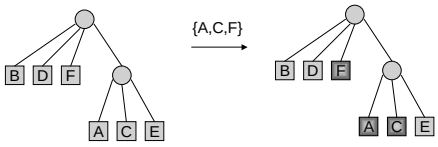
Example (2)

U = {A,B,C,D,E,F}
S = {{A,C,E}, {A,C,F}, {B,D,E}}



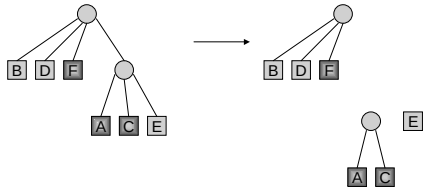
Example (3)

U = {A,B,C,D,E,F}
S = {{A,C,E}, {A,C,F}, {B,D,E}}



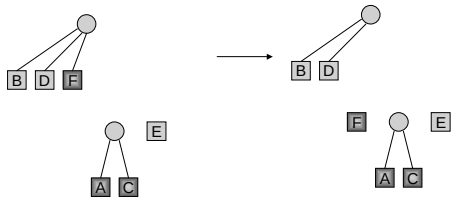
Example (4)

U = {A,B,C,D,E,F}
S = {{A,C,E}, {A,C,F}, {B,D,E}}



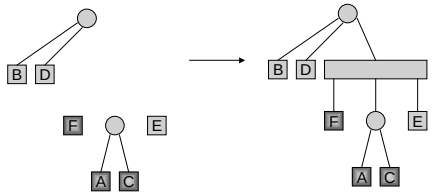
Example (5)

U = {A,B,C,D,E,F}
S = {{A,C,E}, {A,C,F}, {B,D,E}}



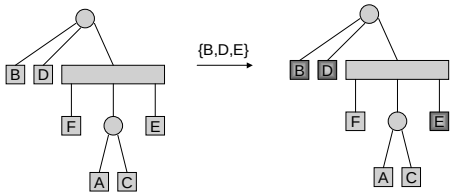
Example (6)

U = {A,B,C,D,E,F}
S = {{A,C,E}, {A,C,F}, {B,D,E}}



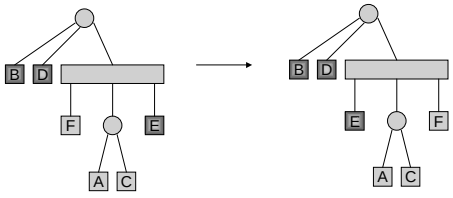
Example (7)

U = {A,B,C,D,E,F}
S = {{A,C,E}, {A,C,F}, {B,D,E}}



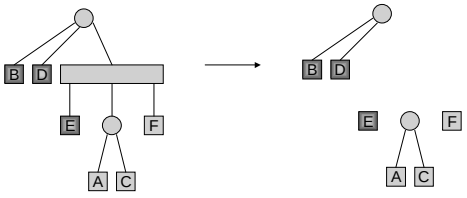
Example (8)

U = {A,B,C,D,E,F}
S = {{A,C,E}, {A,C,F}, {B,D,E}}



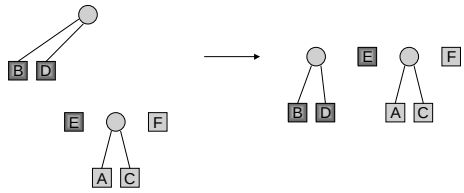
Example (9)

U = {A,B,C,D,E,F}
S = {{A,C,E}, {A,C,F}, {B,D,E}}



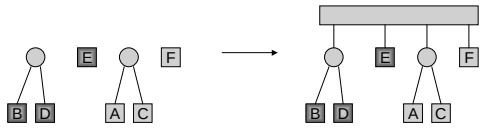
Example (10)

U = {A,B,C,D,E,F}
S = {{A,C,E}, {A,C,F}, {B,D,E}}



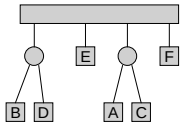
Example (11)

U = {A,B,C,D,E,F}
S = {{A,C,E}, {A,C,F}, {B,D,E}}



Example (12)

U = {A,B,C,D,E,F}
S = {{A,C,E}, {A,C,F}, {B,D,E}}



There are 8 orderings that are possible in keeping each of these sets contiguous.