Computational Biology

Lecture 14

Saad Mneimneh

Physical Mapping

A physical map of a DNA tells the location of precisely defined sequences along the molecule.

- Restriction mapping: mapping of restriction sites of a cutting enzyme based on lengths of fragments
  - Double Digest Problem DDP
  - Partial Digest Problem PDP
- Hybridization mapping: mapping clones based on hybridization data with probes
  - Non-unique probes
  - **Unique probes**

Saad Mneimneh

Unique probes

- Unique probes → each probe occurs only once along the DNA.
- Unique probes are not easy to generate, they are usually long probes.
  - example: STS (Sequence Tag Site) probe is extracted from the DNA itself, often from endpoint of clone, and is sufficiently long that is unlikely to occur a second time on the DNA
- Finding the shortest covering string is this case can be done in polynomial time.

Saad Mneimneh

Shortest covering string

Assuming no hybridization errors

– (1) The length of the shortest covering string is equal to the number of probes.

– (2) The shortest covering string is now given by a special permutation of the probes.

– (1) + (2) => *consecutive ones* property *C*1*P* :

there exists a permutation of the columns of *D*, such that 1's in each row occur in consecutive positions.

Saad Mneimneh

Example of error

- Assume
  - unique probes
  - Obtained the following *D*

i	*j*	*k*
1		1
1	1	
	1	1

- We cannot permute the columns to make *D* satisfy *C*1*P*.

i	*j*	*k*
1		1
1	1	
	1	1

i	*k*	*j*
1	1	
1		1
	1	1

j	*i*	*k*
	1	1
1	1	
1		1

(3 possible permutations up to reversal 3!/2 = 6/2 = 3)

- We must have a hybridization error!
  - possible shortest covering string in this case: *i j k i*

Saad Mneimneh

Finding *C*1*P* permutation

- We will assume no hybridization errors
- The shortest covering string problem reduces to finding a permutation of the columns of *D* to put *D* in *C*1*P* form.
- We will not try to explicitly construct the *C*1*P* permutation, but we will find it by repeatedly identifying neighboring probes.

Saad Mneimneh

Assumption 1

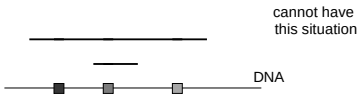
No hybridization errors

(i.e. *C1P* permutation exists)

Saad Mneimneh

Assumption 2

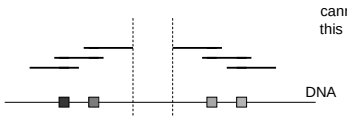
Non-inclusion: No clone *X* contains another clone *Y.*



Saad Mneimneh

Assumption 3

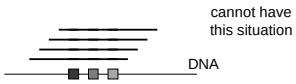
Connectedness: for every partition of the set of probes into two non-empty sets *A* and *B*, there exist probes $i \in A$ and $j \in B$ such that $C_i \cap C_j \neq \phi$



Saad Mneimneh

Assumption 4

Distinguishability: $C_i \neq C_j$ for $i \neq j$



Saad Mneimneh

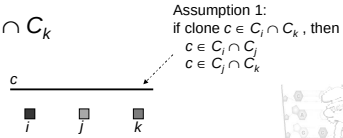
*C*1*P* reformulation (Lemma)

Let 1…*m* be the correct ordering of probes and $1 \leq i < j < k \leq m$. Then

$C_i \cap C_k \subseteq C_i \cap C_j$ and

$C_i \cap C_k \subseteq C_j \cap C_k$

Proof:



Saad Mneimneh

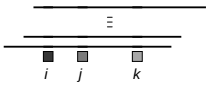
Closest probe

- Given a probe *i* and a set of probes *P* such that $C_i \cap C_j \neq \phi$ for some $j \in P$, a closest probe *k* in *P* to *i* is such that there is no probe in *P* between *i* and *k*.
- If there is only one probe $k \in P$ such that $|C_i \cap C_k|$ is maximized, then there can be no probe between *i* and *k*, and *k* must closest (either from left or from right).
- What if we have a number of probes in *P* such that $|C_i \cap C_k|$ is maximized? In this case, the probe *k* with a minimum $|C_k|$ is closest (either from left or from right).

Saad Mneimneh

Closest probe (cont.)

Consider one side of i (say right) and assume $|C_i \cap C_j| = |C_i \cap C_k| \neq 0$ (this also means that $C_i \cap C_j = C_i \cap C_k$ **no errors**)



C_j and C_k must be different (**distinguishability**)

There must a clone that hybridizes with k but not with j, because we cannot have a clone that hybridizes with j but not with k, otherwise it will be contained in another clone, a contradiction (**non-inclusion**)

Therefore, $|C_j| < |C_k|$

Saad Mneimneh

Strict partial order relation (closer)

j closer than k to i iff

$|C_i \cap C_j| > |C_i \cap C_k|$

or

$|C_i \cap C_j| = |C_i \cap C_k| \neq 0$ and $|C_j| < |C_k|$

Saad Mneimneh

Algorithm

The algorithm maintains an ordered set $\pi = \pi_{first} \ldots \pi_{last}$ denoting a correct sub-sequence of consecutive probes.

compute $|C_i|$ and $|C_i \cap C_j|$ for all probes i and j

$\pi = \pi_{first} = \pi_{last} = i$ for any i
$P = \{$ all probes except $i \}$
repeat
  if $C_{\pi_{last}} \cap C_j = \phi$ for all $j \in P$ [no more probes to the right of π_{last}]
    then reverse π
  choose $j \in P$ with min $|C_j|$ that maximizes $|C_{\pi_{last}} \cap C_j|$
  choose $k \in \{ \pi_{first}, \pi_{last} \}$ with min $|C_k|$ that maximizes $|C_j \cap C_k|$
  if $k = \pi_{last}$
    then $\pi = \pi, j$
    else $\pi = j, \pi$
  $P = P - \{ j \}$
until $P = \phi$

Saad Mneimneh

DNA Sequencing

Saad Mneimneh

DNA sequencing

- To sequence a DNA is to obtain the string of bases that it contains.
- It is impossible to sequence the whole DNA molecule directly.
- We may however obtain a piece of a certain length cut at random and sequence it. This is called a fragment.
- By using cloning and cutting techniques we can obtain a large number of sequenced fragments.
- The goal is to reconstruct the DNA molecule based on the fragments overlap (now the overlap is determined by the explicit sequences).

Saad Mneimneh

Ideal case

- We know the length of the DNA (e.g. ≈ 10 bases)
- There are no errors in sequencing the fragments

```
ACCGT
CGTGC
TTAC
TACCGT
```

```
--ACCGT--
----CGTGC
TTAC-----
-TACCGT--
---------
TTACCGTGC
```

- Align sequences **ignoring end gaps**
- Find consensus by majority voting

Saad Mneimneh

Insertion errors

```
ACCGT          --ACC-GT--
CAGTGC         ----CAGTGC
TTAC           TTAC------
TACCGT         -TACC-GT--
               ----------
               TTACC-GTGC
```

Insertion of A in the second fragment
Gap in consensus will be discarded
In this example, it still works because of majority voting

Saad Mneimneh

Deletion error

```
ACCGT          --ACCGT--
CGTGC          ----CGTGC
TTAC           TTAC-----
TACCGT         -TAC-GT--
               ---------
               TTACCGTGC
```

The first C was deleted from 4th fragment
Consensus still works

Saad Mneimneh

Chimeric fragment

Two disjoint fragments join to form one fragment that is not originally part of the DNA

```
ACCGT          --ACCGT--
CGTGC          ----CGTGC
TTAC           TTAC-----
TACCGT         -TACCGT--
TTATGC         --------
               TTACCGTGC

               TTA---TGC
```

Saad Mneimneh

Unknown orientation

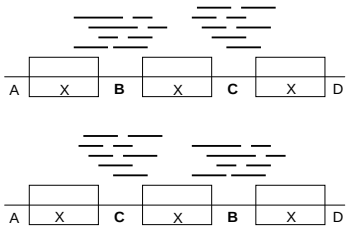
which strand a particular fragment belongs to?

```
CACGT   →  CACGT
ACGT    →  -ACGT          (reverse compliment)
ACTACG  ←  --CGTAGT
GTACT   ←  -----AGTAC
ACTGA   →  --------ACTGA
CTGA    →  ---------CTGA
           ----------
           CACGTAGTACTGA
```

We have 2^n possibilities

Saad Mneimneh

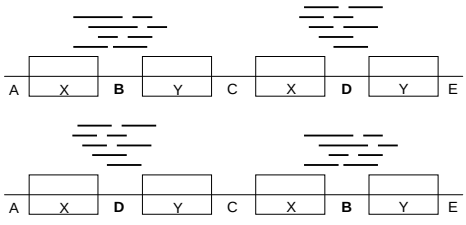
Repeats



Repeats of the form X X X

Saad Mneimneh

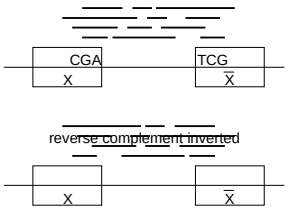
Repeats



Repeats of the form X Y X Y

Saad Mneimneh

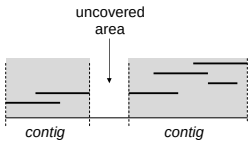
Inverted repeats



Inverted repeat

Saad Mneimneh

Lack of coverage



We have more than one *contig*

Saad Mneimneh

DNA sequencing

- Shortest common superstring SCS
  - "An elegant theoretical abstraction, but fundamentally flawed" – R. Karp

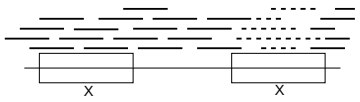
  Given a set of fragments *F*,

  Find the shortest string *s* that contains every $f \in F$ as a substring

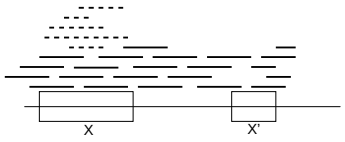
- This is NP-hard
- The SCS might not be what we really want

Saad Mneimneh

Bad example (repeats)



Shortest common superstring will give:



Saad Mneimneh

Solving SCS

We are going to consider a Hamiltonian path approach to solving the SCS problem

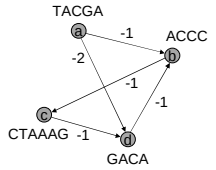
Saad Mneimneh

Overlap graph

- Consider the complete directed weighted graph $G = (V, E)$, called the overlap graph
  - $V = F$ (each fragment is a vertex)
  - $(u,v) \in$ E with weight $-t$ iff t is the length of the maximal suffix of u that is a prefix of v
- We allow self loops and zero weight edges

Saad Mneimneh

Example



a = TACGA
b = ACCC
c = CTAAAG
d = GAGC

0 weight edges
not shown

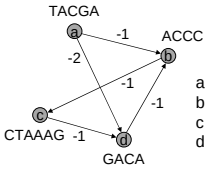
Saad Mneimneh

A path defines a superstring

- Every simple path *P* in the overlap graph involving a set of vertices (fragments) *A* defines a superstring *s*(*P*) for the set *A*.
- Therefore, a Hamiltonian path in the overlap graph defines a superstring for the set of fragments *F*.
- A Hamiltonian path must exist because the graph is complete (how many do we have?).

Saad Mneimneh

Example



a = TACGA
b = ACCC
c = CTAAAG
d = GAGC

0 weight edges
not shown

P = adbc

```
TACGA
    GACA
        ACCC
            CTAAAG
----------------
```

s(*P*): TACGACACCCTAAAG

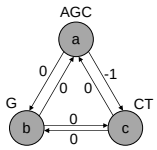
Saad Mneimneh

Does a superstring define a path?

- We have seen that every Hamiltonian path corresponds to a superstring.
- Is the converse true?
  - No: A superstring can contain arbitrary characters that are not present in any fragments
- Does a **shortest** superstring correspond to a Hamiltonian path?
  - Yes: if *F* is substring-free, i.e. no fragment in *F* is contained in another

Saad Mneimneh

Example



The shortest superstring is

AGCT

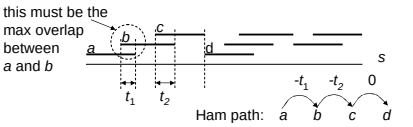
There is no Hamiltonian path *P*, such that *s*(*P*) = AGCT

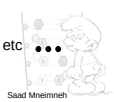
Saad Mneimneh

Subtring-free collection *F*

Let *F* be a substring free set, then for every shortest superstring *s*, there is a Hamiltonian path *P*, such that *s*(*P*) = *s*.

Proof: assume the fragments appear in *s* as follows
(no gaps and no one can be contained in another)





Saad Mneimneh

Non substring-free *F*

- If *F* is not substring-free, then we can remove all fragments from *F* that are substrings of other fragments
- We end up with a set *F*'
- But any superstring of *F*' is a superstring of *F*
- Therefore, we can use *F*'

Saad Mneimneh

Length of string v.s. weight of path

- Let *P* be a Hamiltonian path.
- Let $w(P)$ be the weight of *P*.
- Let $||F|| = \Sigma_{a \in F}|a|$
- Then $|s(P)| = ||F|| + w(P)$ [proof is simple]
- Therefore, the shortest common superstring corresponds to the Hamiltonian path with minimum weight

Saad Mneimneh

Proof

Let *P* be a Hamiltonian path with minimum weight

we need to show that $s(P)$ is a shortest superstring

- Let *s* be a shortest superstring with $|s| < |s(P)|$
- Then there is a Hamiltonian path *P*' such that $s = s(P')$
- $|s(P')| = ||F|| + w(P') < |s(P)| = ||F|| + w(P)$
- Therefore, $w(P') < w(P)$, contradiction

Saad Mneimneh

Hamiltonian path approach

- Finding a minimum weight Hamiltonian path is NP-hard (you can reduce HAMPATH to it)
- Unfortunately, there is no "better" approach to solve SCS, because SCS itself is NP-hard
- Let's consider a greedy algorithm for finding a Hamiltonian path

Saad Mneimneh

Greedy algorithm

- Greedy:
  - start with an empty path
  - repeatedly add the least weighted available edge until you get a Hamiltonian path
- Every time we add an edge (u,v), we need to check:
  - (u,v) does not create a cycle with the previously added edges
  - u has no previously added outgoing edge
  - v has no previously added incoming edge

Saad Mneimneh

Greedy algorithm

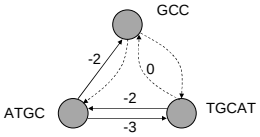
sort edges by their weight: $e_1, e_2, \ldots e_{|E|}$
for all $v \in V$
  in(v) ← 0
  out(v) ← 0
$H \leftarrow \phi$
$i \leftarrow 1$
while $|H| < |F| - 1$
  $(u,v) \leftarrow e_i$
  if out(u) = 0 and in(v) = 0
  then
    if H $\cup$ e_i does not contain a cycle [disjoint set data structure]
      $H \leftarrow H \cup e_i$
      out(u) ← 1
      in(v) ← 1
      $i \leftarrow i + 1$

To build the graph: trivially $O(||F||^2)$
(could be done optimally in
$O(n^2 + ||F||)$ using suffix trees)

To run the algorithm: $O(n^2 \log n)$

Saad Mneimneh

Example



Greedy algorithm will choose:

```
ATGC
  TGCAT
        GCC
ATGCATGCC
```

Optimal is:

```
TGCAT
    ATGC
       GCC
TGCATGCC
```

Saad Mneimneh