Computational Biology

Lecture 13

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

Hybridization Mapping

- Cut several copies of the DNA into overlapping fragments, called clones in this context.
- Construct a set of probes.
- For each clone and probe, determine whether they hybridize.
- From the hybridization data, find an overlap of the clones on the DNA.

Saad Mneimneh

The problem

- Given a hybridization matrix *D*, find the overlap of clones on the DNA (which overlap is the one?)
- Lander-Waterman model
  - Equal length clones at independent random positions
  - Probes placed according to independent Poisson processes
  - Given *D*, find most likely overlap of clones
- Most likely overlap ~ minimize number of occurrences of probes needed to explain *D* [Alizadeh 1995].

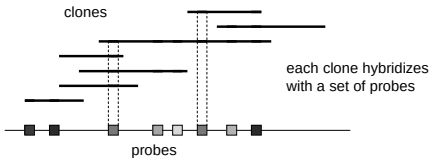
probes

clones	A	B	C	D	E	F	G
1	1		1				
2	1	1			1		
3		1	1	1	1	1	1
4		1	1	1		1	1
5			1	1		1	1
6	1					1	1
7	1	1		1			1
8	1	1		1			1
9				1			

Saad Mneimneh

Covering String

(explains data)



Find a string of probes that can explain the data: a covering string

Saad Mneimneh

Shortest Covering String

- A clone *C* is covered by a string *s* if *s* has a substring containing only the probes of *C* (order and multiplicity ignored).
- Given *D*, find the shortest covering string
  - Non-unique probes (NP-hard)
  - Unique probe (polynomial)

Saad Mneimneh

Non-unique probes

- Non-unique probes → a probe can occur in more than one place along the DNA.
- Non-unique probes are easy to generate.
- Finding the shortest covering string is **NP-hard** with non-unique probes.
- Finding the shortest covering string assuming the clones are covered in a specific order (of their left end points) has a polynomial time algorithm.

Saad Mneimneh

Shortest Covering String (reformulation)

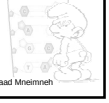
- Let π be a permutation of clones.
- Let s_π be a shortest covering string that covers the clones in the order given by π.
- Shortest covering string = $\min_\pi s_\pi$

Saad Mneimneh

Example

- C_1: {*B*, *C*, *E*}
- C_2: {*A*, *B*, *C*, *D*}
- C_3: {*A*, *B*, *C*}
- C_4: {*B*, *C*, *D*}

- *s* = *A B A C B A C D B C E*
  covers the clones in the order given by the permutation {3, 2, 4, 1}



Saad Mneimneh

Heuristic algorithm (local minimum)

- Start with an arbitrary permutation of clones $\pi = \pi_1 \ldots \pi_n$.
- Let *length* = ∞
- Solve for the shortest covering string s_π in polynomial time — How ?
- let $s = s_\pi$
- While (|s| < *length*)
  - *length* = |s|
  - Compute a set of *neighbor* permutations of π (polynomial number)
  - For each neighbor π' find $s_{\pi'}$ in polynomial time
  - Let *s* be the shortest among these

Saad Mneimneh

Assumptions

- Clones occur in the order (of their left end points) given by a permutation π.
- Motivated by the Lander-Waterman model, all clones have the same length. We will therefore assume that:

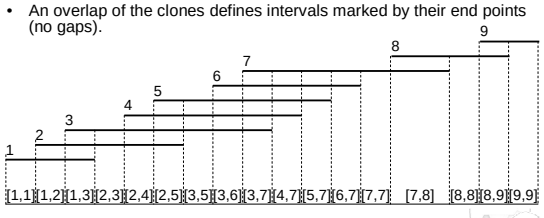
  No clone properly contains another

- We will also assume that the DNA is completely covered by clones; therefore, we have no gaps.

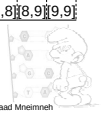
Saad Mneimneh

Intervals

- Consider the clones in order $C_1 \ldots C_n$
- An overlap of the clones defines intervals marked by their end points (no gaps).



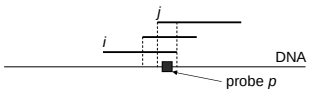
- [*i*,*j*]: clones from C_i to C_j overlap.



Saad Mneimneh

Error-free Hybridization

- For each probe p, and each interval $[i,j]$:
  $p \in (i,j) \Rightarrow p$ hybridizes with all of $C_i, C_{i+1}, \ldots, C_j$

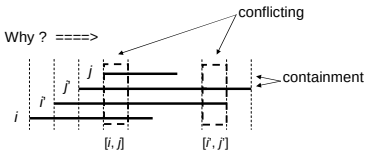


- Therefore, we are considering probes as points.
- We cannot detect overlap smaller than probe length.

Saad Mneimneh

Conflict-free intervals

- The intervals are **conflict-free**
  $[i,j]$ and $[i',j']$ are conflicting if $i < i' \leq j' < j$



- **conflict-free** => we can sort the intervals such that the left end points and the right end points are both sorted.
  - First sort by left end points
  - Then sort by right end points

Saad Mneimneh

overlaps and intervals

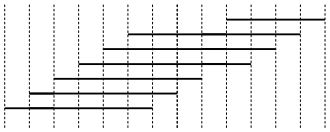
- Every overlap of the clones $C_1 \ldots C_n$ defines a conflict-free set of intervals (trivial).
- For every conflict-free set of intervals I, we have a conflict-free set of intervals I', $I \subseteq I'$, that defines an overlap of the clones $C_1 \ldots C_n$.
  - Add [1,1] and $[n,n]$ to I
  - sort the intervals in I by their left end points and their right end points
  - Fill gaps with new intervals added to I
    $[i,j]$, $[i,j+1] \ldots [i,l]$, $[j+1,l] \ldots [k-1,l]$, **$[k,l]$**

Saad Mneimneh

Example

- $N = 7$
- $I = \{ [4,6], [1,2] \}$

[1,1], [1,2], [1,3], [1,4], [1,5], [1,6], [2,6], [3,6], [4,6], [4,7], [5,7], [6,7], [7,7]



Saad Mneimneh

Example

(there are other ways)

- $N = 7$
- $I = \{ [4,6], [1,2] \}$

[1,1], [1,2], [2,3], [3,4], [4,5], [4,6], [5,6], [6,7], [7,7]

Saad Mneimneh

An attempt

- Given the matrix D, consecutive 1's in a column form a run.
- Consider maximal runs.
- Each run is also an interval.
- If the runs are conflict-free they define some overlap of the clones.
- Make each interval contain its probes.
- We end up with a covering string.

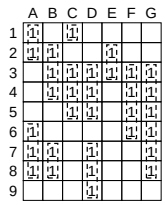
probes

	A	B	C	D	E	F	G
1	1		1				
2	1	1			1		
3		1	1	1	1	1	1
4		1	1	1		1	1
5			1	1		1	1
6	1					1	1
7	1	1		1			1
8	1	1		1			1
9				1			

Saad Mneimneh

Covering String

probes



[1,2], [6,8], [2,4], [7,8], [1,1],
[3,5], [7,9], [2,3], [3,6], [3,8]

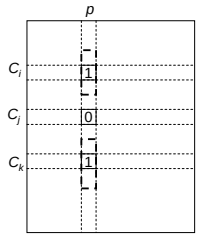
[1,1], [1,2], [2,3], [2,4], [3,5], [3,6], [3,8], [6,8], [7,8], [7,9]
C A E B C D F G A B D

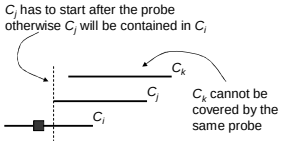
Length of covering string = number of runs

Saad Mneimneh

Is it Shortest ?

Yes… Proof: Each probe has to occur at least a number of times equal to the number of its runs.



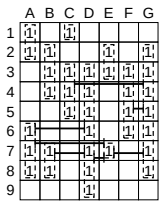


Saad Mneimneh

But…

- What if the runs (intervals) are non conflict-free?
- Some intervals might be conflicting.
- The problem reduces to sub-dividing the set of runs into a conflict-free set with a minimum number of runs.

probes



Saad Mneimneh

Facts

- A set of *t* conflict-free runs defines a covering string of length *t* (we just proved it).
- A covering string of length *t* defines a set of *t* conflict-free runs.
  - Obtain an overlap of the clones $C_1 \ldots C_n$ given by the covering string.
  - The overlap defines conflict-free intervals.
  - Each occurrence of a probe *p* falls within an interval (*i*,*j*) (error-free assumption).
  - An interval [*i*,*j*] containing a probe *p* must be a run in column *p* of matrix *D*, *p* hybridizes with $C_i \ldots C_j$.
  - Therefore, we find *t* such intervals, i.e. *t* conflict-free runs.
- Therefore, obtaining the shortest covering string corresponds to finding the smallest set of conflict-free runs.

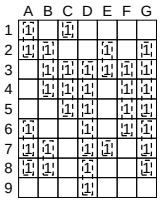
Saad Mneimneh

Obtaining conflict-free runs

Sub-divide the runs such that:

- they are conflict-free
- We minimize the number of runs

probes



[1,1], [1,2], [2,3], [2,4], [2,5], [3,5], [3,6], [3,7], [6,7], [7,7], [7,8], [8,8], [8,9]
C A E B G C F D AG E B AG D

Saad Mneimneh

How to sub-divide runs (intervals)?

Let $[i',j'] \subset_c [i,j]$ mean that $i < i' \leq j' < j$

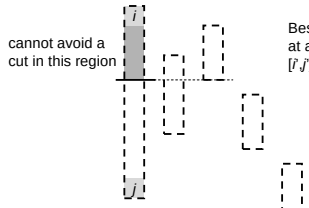
The strategy is as follows:

- Sub-divide an interval [*i*,*j*] into the minimum number of intervals needed to remove any conflict with [*i*',*j*'] where $[i',j'] \subset_c [i,j]$
- The sub-intervals of [*i*,*j*] must not create any additional conflicts (*huh* ?!)
- Do this for every interval
- Therefore, only compare an interval [*i*,*j*] with the original set of intervals

Saad Mneimneh

Looking at one interval [*i*,*j*]

First, let's sub-divide [*i*,*j*] into the minimum number of sub-intervals to remove any conflict with [*i*',*j*'], [*i*',*j*'] $\subset_c$ [*i*,*j*]



cannot avoid a cut in this region

Best thing is to cut at a minimal *j*' such that [*i*',*j*'] $\subset_c$ [*i*,*j*].

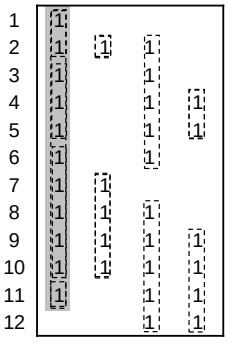
Saad Mneimneh

Sub-dividing [*i*,*j*]

- Find a maximum number of original intervals $[i_1 j_1]$, $[i_2 j_2]$, …, $[i_t, j_t]$ such that:
  - $[i_x, j_x] \subset_c [i,j]$
  - j_1 is minimum,
  - $i_2 > j_1+1$, $i_3 > j_2+1$, …, $i_t > j_{t-1}+1$
- Cut [*i*,*j*] at $j_1, j_2, \ldots, j_t$ producing t+1 sub-intervals
- Each cut is unavoidable and taken as late as possible => this is the minimum number of sub-divisions to remove all conflicts with [*i*',*j*'], [*i*',*j*'] $\subset_c$ [*i*,*j*]

Saad Mneimneh

Example

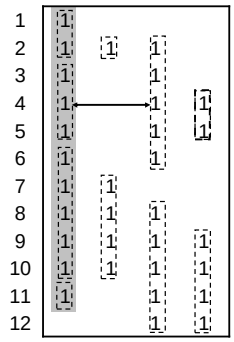


Sub-dividing [1,11] to remove conflicts with [*i*,*j*] such that $1 < i \leq j < 11$

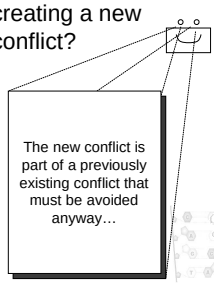
$t = 3$
[2,2],
[4,5],
[7,10]

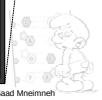
Saad Mneimneh

I am still here !



How can we avoid creating a new conflict?

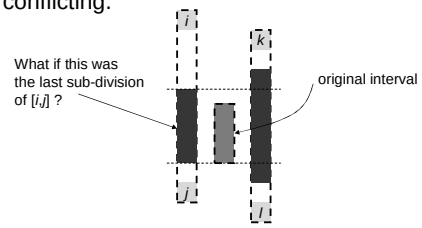




Saad Mneimneh

Let's be formal…

- Assume two sub-divisions of $[i,j]$ and $[k,l]$ are conflicting.

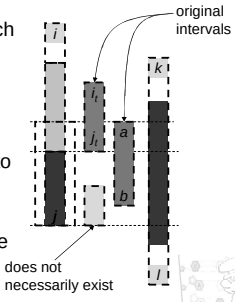


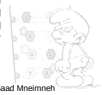
- Then either $[i,j]$ or $[k,l]$ did not resolve all conflicts with original intervals.

Saad Mneimneh

The last sub-division

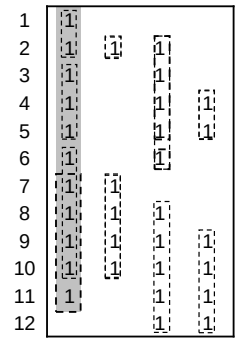
- Consider the interval $[a,b]$ such that $[a,b] \subset_C [i,j]$ and a is maximum such that $a \geq i_t$ (it could be $[i_t,j_t]$ itself).
- Then fix the last sub-interval to be $[a,j]$.
- Then we are back to the same argument.





Saad Mneimneh

Example



Saad Mneimneh