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

Restriction Mapping

- Ordering the fragments maps the restriction sites on the DNA.
- But lengths of fragments is not enough information, any order would satisfy the experiment data.
- Use two cutting enzymes
Double Digest Problem

Double Digest Problem DDP

given
1. multiset $A$ of lengths from enzyme $A$
2. multiset $B$ of lengths from enzyme $B$
3. multiset $C$ of lengths from enzymes $A \times B$

determine an ordering of the fragments that is consistent with the three experiments

Example

- $A = \{3, 6, 8, 10\}$
- $B = \{4, 5, 7, 11\}$
- $C = A \times B = \{1, 2, 3, 5, 6, 7\}$
Number of solutions

- The solution for a DDP might not be unique.
- The number of solutions grows exponentially

Example

- \( \text{Row}_1 \) : \{1, 2, 2, 3, 3, 4\}
- \( \text{Row}_2 \) : \{1, 1, 2, 2, 4, 5\}
- \( \text{Row}_3 \) : \{1, 1, 1, 1, 2, 3, 3\}

A
\[
\begin{array}{cccccc}
3 & 2 & 1 & 4 & 5 & 6 \\
1 & 3 & 2 & 1 & 4 & 5 \\
5 & 2 & 3 & 1 & 4 & 6 \\
\end{array}
\]

B
\[
\begin{array}{cccccc}
3 & 2 & 1 & 4 & 5 & 6 \\
2 & 3 & 1 & 4 & 5 & 6 \\
3 & 1 & 4 & 5 & 6 & 2 \\
\end{array}
\]

C = A \times B
\[
\begin{array}{cccccc}
1 & 3 & 2 & 4 & 5 & 6 \\
3 & 1 & 4 & 5 & 6 & 2 \\
2 & 3 & 1 & 4 & 5 & 6 \\
\end{array}
\]

Equivalence of Solutions

- Some different solutions might be equivalent.
- For instance, if \((a_1, a_2, \ldots, a_m) (b_1, b_2, \ldots, b_n)\) is a solution, then \((a_m, a_{m-1}, \ldots, a_1) (b_n, b_{n-1}, \ldots, b_1)\) is also a solution.
- This is a Reflection. No fragment length data could possibly distinguish between the two, they only differ by orientation.
Reflection

\[
\begin{array}{c|cccc}
\text{A} & 1 & 2 & 4 & 5 \\
\hline
\text{B} & 5 & 1 & 2 & 3 \\
\text{C} = \text{A} \times \text{B} & 7 & 11 & 5 & 4
\end{array}
\]

Overlap Equivalence

- Let’s define a more general type of equivalence called overlap equivalence.
- Let \([A]\) be the set of fragments from \(A\) and \([B]\) be the set of fragments from \(B\).
- A solution defines an overlap matrix \(O\),
  s.t. \(O_{ij} = 1\) if \(A_i\) overlaps with \(B_j\).
- Two solutions are overlap equivalent if they define the same overlap matrix \(O\).
- Reflections are overlap equivalent.

Equivalence class

- A solution with all its overlap equivalent solutions form an equivalence class (this is an equivalence relation).

- Given a solution,
  - What is the size of its equivalence class?
  - Can we generate all solutions in the class?
Observations

- If a solution has $t - 1$ coincident cuts sites, then it has $t$ components.
- They can be permuted in $t!$ ways without changing the overlap data.
- Each component can also be reflected without changing the overlap.
- Therefore, we can generate $2^t!$ solutions.

Example

Another observation

- Given a solution, let
  - $\mathcal{S}_1 = \{ A_i : A_i \subset B_j \}$
  - $\mathcal{S}_2 = \{ B_k : B_k \subset A_i \}$
- Permuting $\mathcal{S}_1$ and $\mathcal{S}_2$ does not change the overlap data
Example

\[ A \times B \]

Size of equivalence class

- Is it \(2\pi \prod |s_j| \Pi |\sigma_j|\)?
- Not quite!
- If a component has only one fragment in either \(A\) or \(B\), then a reflection is also a permutation.
- Let \(s\) be the number of such components, then the size of the equivalence class is:
  \[2^{s-\pi} \prod |s_j| \Pi |\sigma_j|\]

Other Equivalences?

- We can define other kinds of equivalences.
- Consider overlap size equivalence, i.e. two solutions are equivalent if they produce the same overlap sizes.
- Overlap equivalence => overlap size equivalence, but not the other way around.
Example

\[ \text{switch} \implies \text{overlap size equivalent} \]

\[ A \cap C = A \wedge B \]

\[ A \setminus \text{overlaps with } B \]

Cassette Transformation Equivalence

- Let \(|C| = l\)
- For \(1 \leq i \leq j \leq l\), \(I_C = \{ C_k : i \leq k \leq j \}\) is the set of fragments from \(C_i\) to \(C_j\).
- The cassette defined by \(I_C\) is the set of fragments \((I_A, I_B)\) that contain a fragment from \(I_C\).

Cassette

Cassette for \(L = \{ C_1, C_2, C_3, C_4, C_5 \}\)
Cassette left and right overlap

left overlap = \( m_1 - m_0 \)
right overlap = \( m_2 - m_1 \)

Cassette exchange

Left overlap = -2
Right overlap = 4

Two cassettes with the same left and right overlap can be exchanged

Cassette Reflection

Left overlap = -1
Right overlap = 1

A cassette with the same left and right overlap (but different signs) can be reflected
Cassette Equivalence

- Two solutions are cassette equivalent if there exists a series of cassette transformations (exchanges and reflections) that take on to the other.

- What is the size of an equivalence class?

Alternating Euler Paths

- Consider a graph with colored edges

- An Euler path (cycle) is a path (cycle) that goes through every edge once

- An alternating Euler path (cycle) is an Euler path (cycle) such that consecutive edges on the path (cycle) have different colors

- Pevzner 1995 showed that given a solution, we can construct a special bi-colored graph called the border block graph.

- Each cassette equivalent solution corresponds to an alternating Euler path (cycle) in the graph and vice-versa.

Fact

- Let $d_i(v)$ be the number of edges of color $i$ incident to $v$.

- An edge bi-colored connected graph with $d_a(v) = d_b(v)$ has an alternating Euler cycle.

- Proof:
  - Every vertex has even degree; therefore, the graph contains an Euler cycle.
  - Construct the Euler cycle the usual way, but by using only distinct color edges when traversing a vertex.
Exchange

- Consider an alternating path 
  $...x...y...x...y...$

- It consists of 5 parts $F_1F_2F_3F_4F_5$

- $F_1F_2F_3F_4F_5 \Rightarrow F_1F_4F_3F_2F_5$ is called an exchange if $F_1F_4F_3F_2F_5$ is an alternating path

Illustration

Reflection

- Consider an alternating path 
  $...x...x...$

- It consists of 3 parts $F_1F_2F_3$

- $F_1F_2F_3 \Rightarrow F_1F_2^2F_3$ is called a reflection if $F_1F_2^2F_3$ is an alternating path, where $F_2^2$ is the reverse of $F_2$. 
Fact

• Every two alternating Euler cycles in a bi-colored graph can be transformed into each other by a series of exchanges and reflections.

• Proof: Pevzner p. 29

The border blocks

• Let \( I(A_i) = \{ C_k : C_k \subset A_i \} \)

• Let \( I(B_j) = \{ C_k : C_k \subset B_j \} \)

• If \( |I(X)| > 1 \), define the border blocks of \( X \) to be the left most and right most block in \( I(X) \).

• \( C_i \) is a border block if it is a border block for some fragment \( X \).
Example

\[ \mathcal{K}(A) = \{ C_1, C_2, C_3, C_4 \} \]
Border blocks of \( A_1, C_2 \) and \( C_3 \)

Lemma

- Each fragment \( X \) with \( |k(X)| > 1 \) contains exactly two border blocks.
- \( k(A) \cap k(B) \leq 1 \)
- Assume no cuts in \( A \) and \( B \) coincide, then each border block is a border block for some \( A_i \) and some \( B_j \), except \( C_1 \) and \( C_2 \).

Border graph

- Let \( \mathcal{S} = \{ C_0 : C_0 \text{ is a border block} \} \)
- \( V = \{ |C_i| : C_0 \in \mathcal{S} \} \) vertices
- \( E = \{ (|C_i|, |C_j|) : C_i, C_j \in \mathcal{S} \cap k(X) \text{ for some } X \} \)
- Each edge labeled by its \( X \) and colored \( A \) if \( X \in A \) and \( B \) if \( X \in B \).
Example

Alternating Euler path in border block graph

- Each vertex has equal number of edges of each color, except possibly for $|C_1|$ and $|C_2|$.
- By adding one or two edges (depending on the colors) we can fix this. Therefore, the graph has an alternating Euler path or cycle.
- Let $C_1, \ldots, C_n$ be the ordered set of border blocks, then $\tilde{P} = [C_1, \ldots, C_n]$ is an alternating Euler path (cycle).

Result

Cassette transformations do not change the border graph.

Let $P$ be the alternating Euler path (cycle) corresponding a solution $[A, B]$.

- If a solution $[A', B']$ is obtained from $[A, B]$ by cassette exchange (reflection), it will have a path $P'$ that can be obtained from $\tilde{P}$ by an exchange (reflection).
- Let $P'$ be an alternating Euler path (cycle) obtained from $P$ by exchange (reflection). Then there is a solution $[A', B']$ that can be obtained from $[A, B]$ by cassette exchange (reflection), where $P'$ corresponds to $[A', B']$. 
Example

This corresponds to the cycle:

$B_1 A_1 B_2 A_2 A_1 B_1$  
$1 1 2 2 1 2 1$

Example (cont.)

This corresponds to the cycle:

$B_1 A_1 B_2 A_2 A_1 B_1$  
$1 1 2 2 1 2 1$

Example

Cycle 1:

$B_1 A_1 B_2 A_2 A_1 B_1$  
$1 1 2 2 1 2 1$

Cycle 2:

$B_1 A_1 B_2 A_2 A_1 B_1$  
$1 1 2 2 1 2 1$
DDP is NP-complete

Proof:
- $DPP \in \text{NP}$. A solution for $DPP$ can be verified in polynomial time.
- Set Partition problem (classical one), which is NP-complete, reduces to $DPP$ in polynomial time.

$DPP \in \text{NP}$

- Given
  - multiset $A$ of lengths from enzyme $A$
  - multiset $B$ of lengths from enzyme $B$
  - multiset $C$ of lengths from enzymes $A + B$
- Solution
  - two sets of restriction sites, $a$ and $b$.
- Verification:
  - Sort $a + b = (0, l)$. $L =$ sum of all lengths in $A$
  - Compute multiset $c = \{ c_i; c_i = a_i - g_i, 0 \leq i \leq l \}$ and $b_i = g_i$
  - Sort $c$ and $C$ and compare them

Set Partition

Set Partition $SP$:

Given a set $J$ of integers with total sum $J$

Can we partition $J$ into two sets of sum $J/2$ each?

is NP-complete.

Given an $SP$ instance, construct the following DDP:

- $A = J$
- $B = \{ J/2, J/2 \}$
- $C = J$

$DPP$ has a solution iff $SP$ has a solution
Partial Digest

- One restriction enzyme only
- Obtain fragments for every pair of cuts and every cut and a boundary
- This can be achieved by missing a number of sites in every trial

Partial Digest Problem (turnpike problem)

**Partial Digest Problem PDP**

given

- multiset \( \{x \} \) of distances between every pair of points on the line

\[
|\mathcal{X}| = \binom{n}{2} - \binom{n-2}{2}
\]

reconstruct the points on the line

PDP

- No polynomial time algorithm known
- Not known to be NP-complete
- Practical Backtracking algorithm due to Skiena et al. 1990
The algorithm

- Find longest distance in ΔX, this decides the two outermost points, delete that distance from ΔX.
- Repeatedly position the longest remaining distance of ΔX.
- Since the longer distance must be realized from one of the two outermost points, we have two possible positions (left and right) for the point.
- For each of these two positions, check whether all the distances from the position to the points already positioned are in ΔX.
- If they are, delete all those distances from ΔX and proceed.
- Backtrack if they are not for both of the two positions.

Example

ΔX = (2, 2, 3, 4, 5, 6, 7, 8, 10)

ΔX = (2, 2, 3, 4, 5, 6, 7, 8, 10)

ΔX = (2, 2, 3, 4, 5, 6, 7, 8, 10)

Example (cont.)

ΔX = (2, 2, 3, 4, 5, 6, 7, 8, 10)

ΔX = (2, 2, 3, 4, 5, 6, 7, 8, 10)
Worst case: Exponential

We could end up backtracking an exponential number of times in this decision tree.

It is not trivial to show that an exponential number of backtracking could occur.

Zhang (1994) provided an example.

In practice...

• If we have real points in general positions, then one of the two choices will be pruned with probability 1.

• The algorithm runs in $O(n^2 \log n)$ expected time since we have $O(n^2)$ positions and all operations can be done on $\Delta X$ using binary search which takes $O(\log n^2) = O(\log n)$.

In practice…

If we have real points in general positions, then one of the two choices will be pruned with probability 1.

The algorithm runs in $O(n^2 \log n)$ expected time since we have $O(n^2)$ positions and all operations can be done on $\Delta X$ using binary search which takes $O(\log n^2) = O(\log n)$. 

In practice…

If we have real points in general positions, then one of the two choices will be pruned with probability 1.

The algorithm runs in $O(n^2 \log n)$ expected time since we have $O(n^2)$ positions and all operations can be done on $\Delta X$ using binary search which takes $O(\log n^2) = O(\log n)$. 

In practice…

If we have real points in general positions, then one of the two choices will be pruned with probability 1.

The algorithm runs in $O(n^2 \log n)$ expected time since we have $O(n^2)$ positions and all operations can be done on $\Delta X$ using binary search which takes $O(\log n^2) = O(\log n)$. 

In practice…

If we have real points in general positions, then one of the two choices will be pruned with probability 1.

The algorithm runs in $O(n^2 \log n)$ expected time since we have $O(n^2)$ positions and all operations can be done on $\Delta X$ using binary search which takes $O(\log n^2) = O(\log n)$. 

In practice…

If we have real points in general positions, then one of the two choices will be pruned with probability 1.

The algorithm runs in $O(n^2 \log n)$ expected time since we have $O(n^2)$ positions and all operations can be done on $\Delta X$ using binary search which takes $O(\log n^2) = O(\log n)$. 

In practice…

If we have real points in general positions, then one of the two choices will be pruned with probability 1.

The algorithm runs in $O(n^2 \log n)$ expected time since we have $O(n^2)$ positions and all operations can be done on $\Delta X$ using binary search which takes $O(\log n^2) = O(\log n)$. 

In practice…

If we have real points in general positions, then one of the two choices will be pruned with probability 1.

The algorithm runs in $O(n^2 \log n)$ expected time since we have $O(n^2)$ positions and all operations can be done on $\Delta X$ using binary search which takes $O(\log n^2) = O(\log n)$. 

In practice…

If we have real points in general positions, then one of the two choices will be pruned with probability 1.

The algorithm runs in $O(n^2 \log n)$ expected time since we have $O(n^2)$ positions and all operations can be done on $\Delta X$ using binary search which takes $O(\log n^2) = O(\log n)$. 

In practice…

If we have real points in general positions, then one of the two choices will be pruned with probability 1.

The algorithm runs in $O(n^2 \log n)$ expected time since we have $O(n^2)$ positions and all operations can be done on $\Delta X$ using binary search which takes $O(\log n^2) = O(\log n)$. 

In practice…

If we have real points in general positions, then one of the two choices will be pruned with probability 1.

The algorithm runs in $O(n^2 \log n)$ expected time since we have $O(n^2)$ positions and all operations can be done on $\Delta X$ using binary search which takes $O(\log n^2) = O(\log n)$. 

In practice…

If we have real points in general positions, then one of the two choices will be pruned with probability 1.

The algorithm runs in $O(n^2 \log n)$ expected time since we have $O(n^2)$ positions and all operations can be done on $\Delta X$ using binary search which takes $O(\log n^2) = O(\log n)$. 

In practice…

If we have real points in general positions, then one of the two choices will be pruned with probability 1.

The algorithm runs in $O(n^2 \log n)$ expected time since we have $O(n^2)$ positions and all operations can be done on $\Delta X$ using binary search which takes $O(\log n^2) = O(\log n)$.
Example

- $A$: \{1, 2, 3, 5, 6\}
- $B$: \{1, 3, 4, 9\}
- $C = A \lor B$: \{1, 1, 1, 2, 3, 3\}

\[
\begin{array}{ccc|ccc}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
A & 1 & 1 & 1 & 0 & 0 & 0 \\
B & 1 & 1 & 1 & 0 & 0 & 0 \\
C = A \lor B & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]