Computational Biology (MAT4930/MAT5932)

Lecture 3: Exhaustive Search

Abdulmelik Mohammed

University of South Florida

01/19/2021

A DNA-cleaving enzyme from *Haemophilus influenzae*

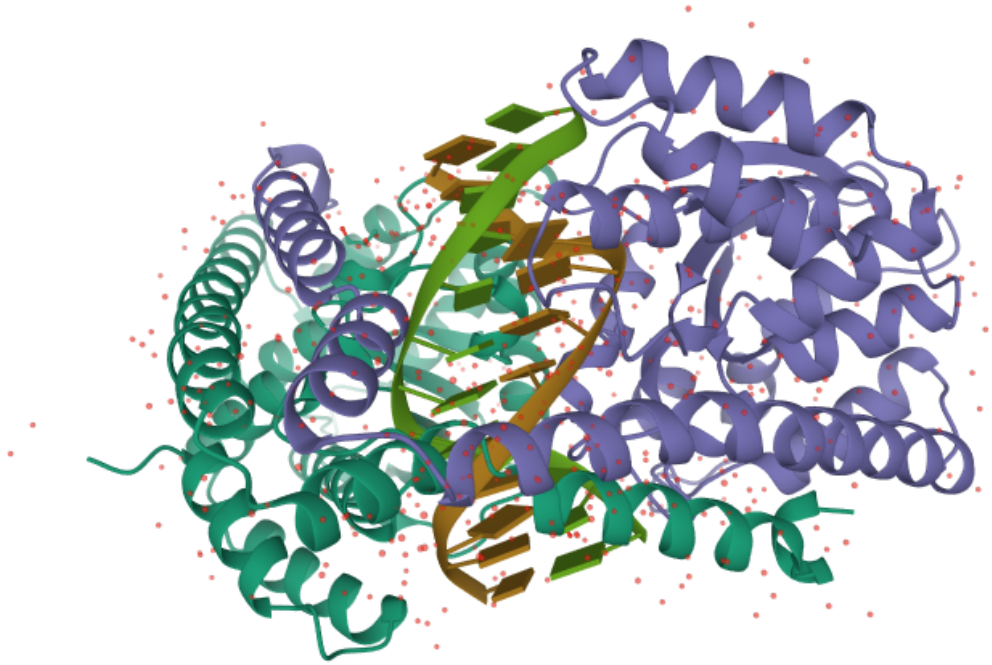


Figure: HindIII restriction enzyme. PDB structure 2E52 (`http://www.rcsb.org/3d-view/2E52`).

Restriction enzymes

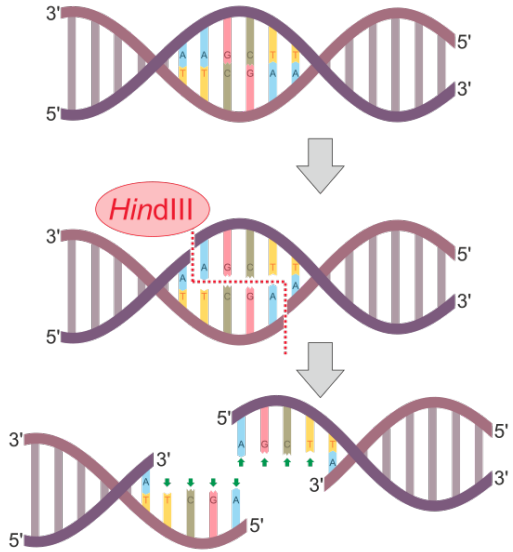


Figure: HindIII cutting DNA. Image by Helixitta 2015.

- A *restriction enzyme* cleaves a DNA molecule at a specific sequence, breaking it into a collection of *restriction fragments*.

Restriction mapping

- *Restriction mapping* is the process of identifying the location of restriction sites in a long DNA.
- Before sequencing technologies, biologists built restriction maps through experimental techniques such as gel electrophoresis.
- The distance between restriction sites corresponds to the length of restriction fragments.
- The fragments can be separated by length using gel electrophoresis.

Gel electrophoresis

- In gel electrophoresis, DNA samples are loaded into lanes in the gel box.
- The electric field in the gel drives the DNA samples along the lanes of the gel.

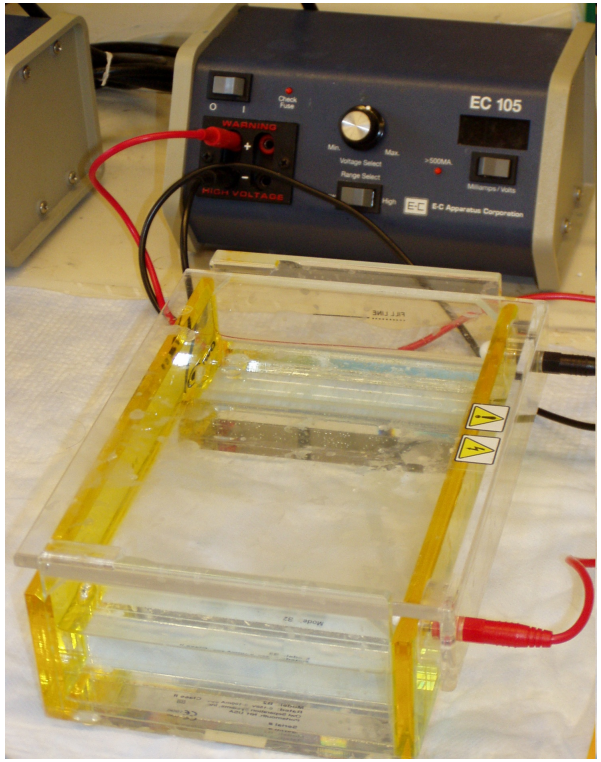


Figure: Gel electrophoresis device. Image by J. M. Vinocur 2006.

Gel electrophoresis separation

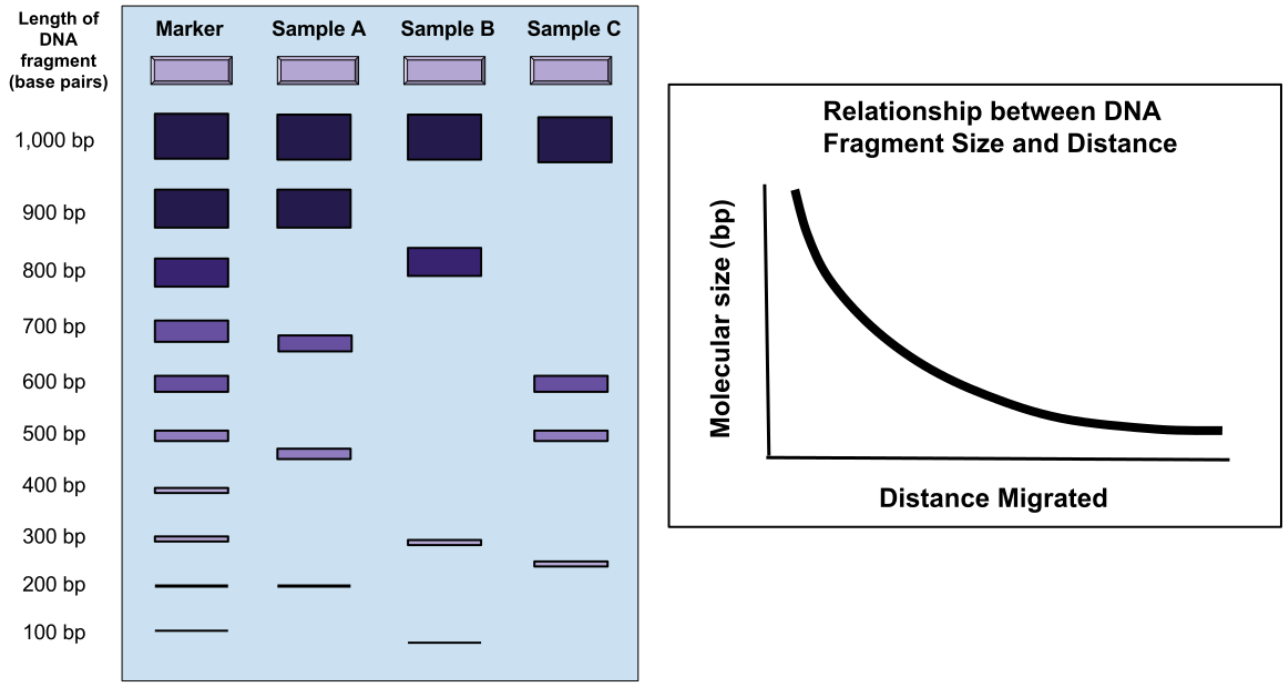
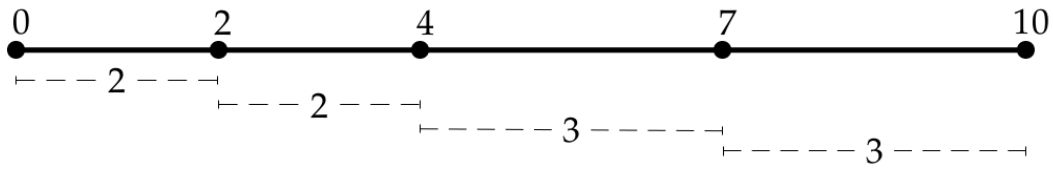
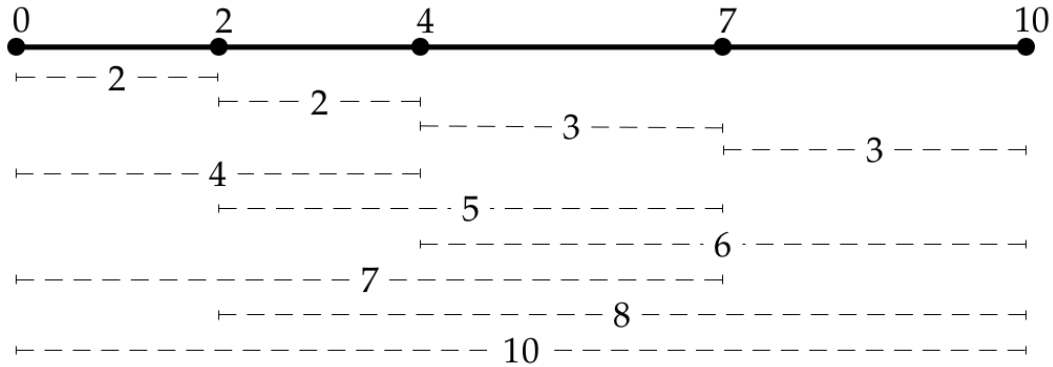


Figure: Separating DNA by length using gel electrophoresis. Image by Mckenzielower 2017.

Experimentalists can control conditions to get complete or partial digests



(a) Complete digest.



(b) Partial digest.

Figure: Complete vs partial digests. Dots indicate the restriction sites. Image reprinted from Jones and Pevzner 2004.

Computational modeling of restriction mapping: Partial Digest Problem (PDP)

- Goal: recover location of sites from lengths of fragments.
- Model of restriction sites: a set $X = \{x_1 = 0, x_2, \ldots, x_n\}$ of integer points in the real line in increasing order.
- Model of distances between restriction sites: the multiset of all $\binom{n}{2}$ possible distances ΔX between points in X; i.e. $\Delta X = \{x_j - x_i : 1 \leq i < j \leq n\}$.
- PDP problem: reconstruct X from ΔX. I.e. input $= \Delta X$ and output $= X$.

Example

- Take $X = \{0, 2, 4, 7, 10\}$. Then $\Delta X = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$.

	0	2	4	7	10
0		2	4	7	10
2			2	5	8
4				3	6
7					3
10					

Table: ΔX as a table. Element (i, j) in the table shows $x_j - x_i$.

Non-unique solutions

Remark

It may not be possible to uniquely reconstruct X from ΔX.

Example

$\Delta X = \Delta(X + v)$, where v is any positive integer and $X + v = \{x + v : x \in X\}$ a.

- Other more interesting examples of distinct X_1 and X_2 with $\Delta X_1 = \Delta X_2$ exist.
- It can be of interest to generate all possible solutions.

A brute force algorithm for PDP

Input: A multiset L of $\binom{n}{2}$ pairwise integer distances.
Output: A set X of n non-negative integers that includes 0 and with $\Delta X = L$.

```
1  M ← maximum element in L;
2  X ← ∅, # return empty set if no solution;
3  foreach set of n − 2 integers 0 < x_2 < ... < x_{n−1} < M do
4  |   X ← {0, x_2, ..., x_{n−1}, M};
5  |   Y ← Δ(X), # Details omitted;
6  |   if Y = L then
7  |   |   return X;
8  |   end
9  end
10 return X;
```

Algorithm 1: BruteForcePDP algorithm.

Analysis

- X cannot contain a number x greater than M; otherwise $x - 0 = x > M$ would be in L and thus M could not have been a maximum.
- The algorithm tries $\binom{M-1}{n-2}$ possible combinations for X.
- Assuming for simplicity that Line 5 takes $O(1)$ time, this is roughly $O(M^{n-2})$ time. (In fact, Line 5 takes longer.) Can we do better?
- Note that each $x \neq 0$ in a proper solution X must be contained in L as $x - 0$ is in ΔX.
- Thus there is no need to check sets which contain an element outside L!

A better brute force search algorithm for PDP

Input: A multiset L of $\binom{n}{2}$ pairwise integer distances.
Output: A set X of n non-negative integers that includes 0 and with $\Delta X = L$.

```
1  M ← maximum element in L;
2  X ← ∅, # return empty set if no solution;
3  foreach set of n − 2 integers 0 < x_2 < ... < x_{n−1} < M from L
     do
4      X ← {0, x_2, ..., x_{n−1}, M};
5      Y ← Δ(X), # Details omitted;
6      if Y = L then
7          return X;
8      end
9  end
10 return X;
```

Algorithm 2: BetterBruteEorcePDP algorithm.

Analysis

- BetterBruteEorcePDP tries $\binom{|L|}{n-2}$ possible sets.
- $|L| = \frac{n(n-1)}{2}$ and thus the algorithms running time is roughly $O(n^{2n-4})$.
- The running time does not depend on M!
- Compare the two algorithms on input $L = \{2, 998, 1000\}$.
- Nevertheless, BetterBruteEorcePDP is still not a practical algorithm.

Backtracking and branch-and-bound algorithms

- In both of the brute force algorithms, we went through the possible space of complete solutions.
- A common technique to improve performance is to build candidate solutions piece by piece and eliminating partial solutions which cannot lead to a proper full solution.
- *Backtracking* and *brach-and-bound* algorithms follow this principle.

A practical backtracking algorithm for restriction mapping

- For the PDP problem, note that the largest value M_1 in L is determined by the two outermost values $\{0, M_1\}$ in $X = \{0, x_2, \ldots, x_{n-1}, M_1\}$.
- Thus, the partial solution $\{0, M_1\}$ can serve as a starting point.
- We can proceed by considering the next largest distance M_2 in L. Note that $M_2 = x_{n-1} - 0$ or $M_2 = M_1 - x_2$. We can thus explore the two possible extended partial solutions $\{0, M_1 - M_2, M\}$ and $\{0, M_2, M_1\}$.

- Let us consider one of the extensions $\{0, M_1 - M_2, M_1\}$. In this case, both $M_1 - M_2 - 0$ and $M_1 - (M_1 - M_2) = M_2$ must be in L. If either one is not in L, we can discard this partial solution and *backtrack*. Otherwise, we can continue extending the partial solution to see if it eventually becomes a viable complete solution.
- Note that, in the next extension, we ignore the elements of L that have been addressed by the current partial solution. That is, we must consider the largest element in $L \setminus \{M_1, M_2, M_1 - M_2\}$.
- If after trying different extensions (and backtracking as necessary) we reach a state where all the elements of L are addressed, we have found a complete solution that we can output. Otherwise, there is no solution.

Example $L = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$

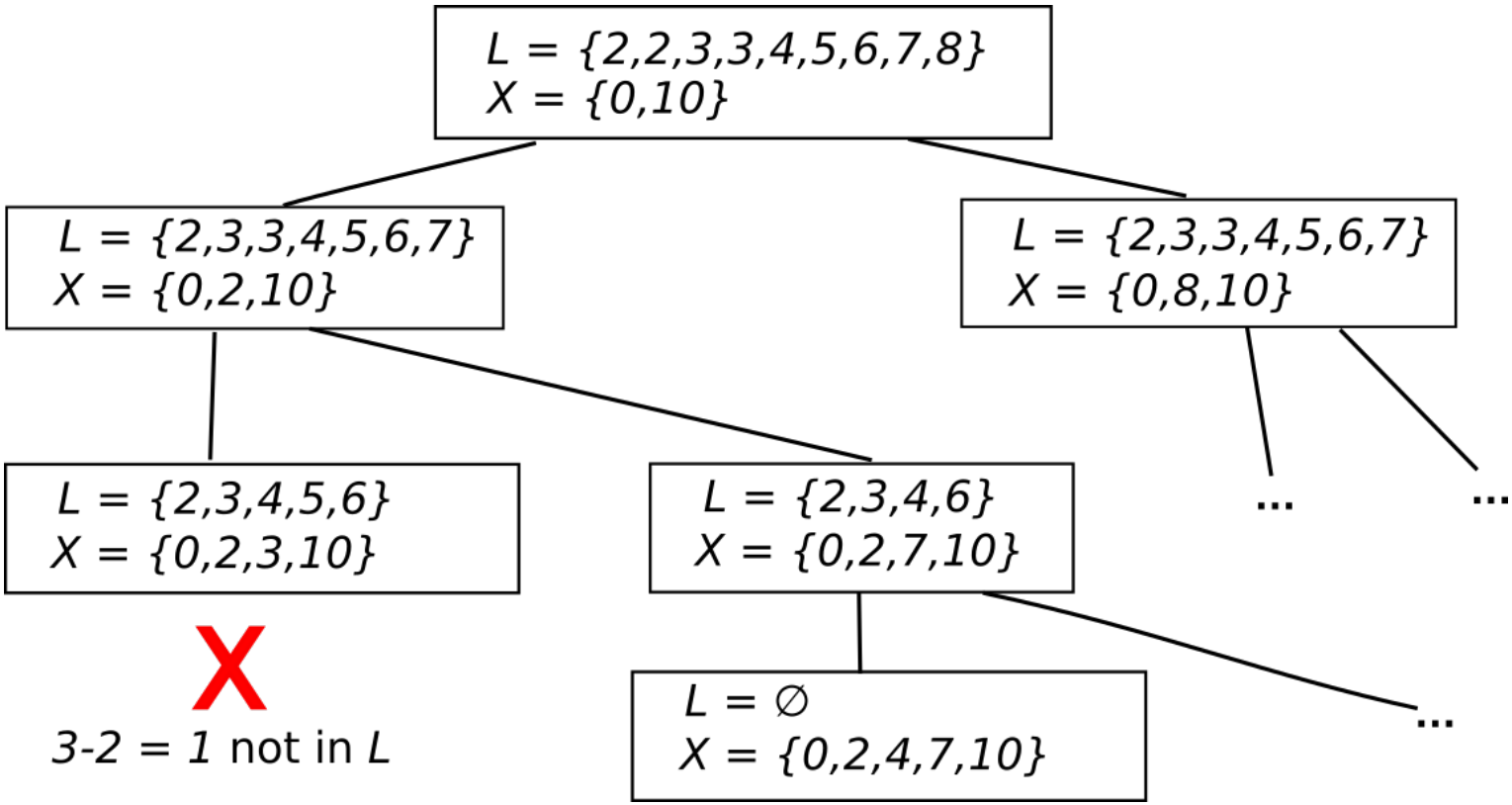


Figure: Backtracking search tree for partial digest algorithm

Partial digest algorithm

- To state the algorithm we introduce a few additional notations and subroutines.
- Let $\Delta(y, X)$ denote the multiset of distances between y and all the points in X. For e.g. $\Delta(2, \{1, 3, 4, 5\}) = \{1, 1, 2, 3\}$.
- $\textsc{Delete}(y, L)$ is a subroutine that deletes y from L.
- We implement the backtracking through a recursive algorithm.

Partial digest algorithm

```
1 PartialDigest(L)
2 |   width ← maximum element in L
3 |   Delete(width, L)
4 |   X ← {0, width}
5 |   Place (L, X)
6 |   end
```

Algorithm 3: PartialDigest algorithm.

Partial digest algorithm: PLACE subroutine

```
1  PLACE(L, X)
2  |  if L = ∅ then
3  |  |  Output X
4  |  end
5  y ← maximum element in L
6  if Δ(width − y, X) ⊆ L then
7  |  Add width − y to X and remove lengths Δ(width − y, X)
   |   from L
8  |  PLACE (L, X)
9  |  Remove width − y from X and add lengths
   |   Δ(width − y, X) to L
10 end
11 if Δ(y, X) ⊆ L then
12 |  Add y to X and remove lengths Δ(y, X) from L
13 |  PLACE (L, X)
14 |  Remove y from X and add lengths Δ(y, X) to L
15 end
16 return
```

Analysis

- From the example, we can tell that the search tree is binary and has depth at most n, where n is the size of X.
- Hence, the run-time of the algorithm is $O(2^n)$.
- There exist problem instances where the algorithm runs in $\Omega(2^n)$ [2] and hence the analysis is tight.
- There exists a polynomial time algorithm for the PDP problem [1].

References


A. Daurata, Y. Gérardb, M. Nivatc, The chords' problem, *Theoretical Computer Science* 282:319–336, 2002.


Z. Zhang, An exponential example for a partial digest mapping algorithm, *Journal of Computational Biology*:235–239, 1994.