Computational Biology
Lecture 15: DNA sequencing and the shortest superstring problem
Saad Mneimneh

To sequence a DNA is to obtain the sequences of bases it contains. It is almost impossible to sequence the whole molecule directly. We may, however, obtain a fragment of a relatively small length that is cut at random from the DNA molecule, and sequence it. 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.

The shortest superstring problem

A naive abstraction for the sequencing problem is the Shortest Superstring problem stated below:

Shortest Superstring Problem: Given a set of sequenced fragments F, find a shortest string s that contains every $f \in F$ as a substring.

The shortest superstring problem is:

"An elegant theoretical abstraction, but fundamentally flawed"

Richard Karp at CSB2003, Stanford.

The above statement was made in the context of illustrating how computer scientists and biologists seek different things. When the problem of sequencing arised, theoreticians came up with the above abstraction of the shortest superstring problem (an NP-hard one), and started to work on efficient approximation algorithms for it, while the shortest superstring was not what biologists really wanted!

The shortest superstring does not model possible errors arising from sequencing the individual fragments. Errors might include insertion errors where a base (or more) is wrongly present in a fragment, deletion errors where a base (or more) is absent from a fragment, substitution errors where a base (or more) is substituted for another in a fragment, chimeric errors where two disjoint fragments join to form a single non-existing fragment, etc... Moreover, the shortest superstring does not model fragment orientation since the source (unknown) of a fragment can be one of the two strands of the DNA.

Most importantly, the shortest superstring abstraction fails in the presence of repeats. Suppose that the DNA molecule has two copies of an exact repeat and that fragments are smapled as shown below:

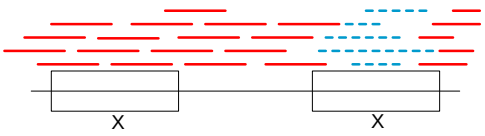


Figure 1: Two copies of X

Note that the repeat copies are long and contain many fragments. In this case, even if the fragments are exact substrings of the DNA (no errors) and even if we know their correct orientation, finding the shortest superstring may not be what we want.

For instance, the following figure shows a different assembly with a shortest DNA molecule for the same set of fragments. Because the repeat copies are identical, a superstring may contain only one copy of X, which will absorb all fragments totally contained in any of the copies. The other copy X' will be shorter and will contain only the fragments that cross the border of X.

Therefore, the shortest superstring may result in a shorter DNA and ignore all repeats.

Nevertheless, the shortest superstring is an interesting theoretical problem. It is NP-hard but approximation algorithms exist. Given all its shortcomings, such algorithms are primarily of theoretical interest. But we will characterize the shortest superstring as a Hamiltonian path in a directed graph which will motivate an efficient (polynomial time) approach for another biological problem known as Seuqncing By Hybridization.

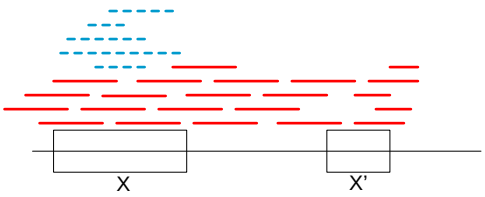


Figure 2: One copy of X, shorter DNA

Shortet superstring as Hamiltonian path

Consider the complete directed weighted graph $G = (V, E)$, called the overlap graph, defined as follows:

- $V = F$ (a vertex for every fragment)
- $(u, v) \in E$ with a weight $w(u, v) = -t$ iff $t \geq 0$ is the length of the maximal suffix of u that is a prefix of v

Therefore, we allow self loops and zero weight edges in G.
Here's an example:

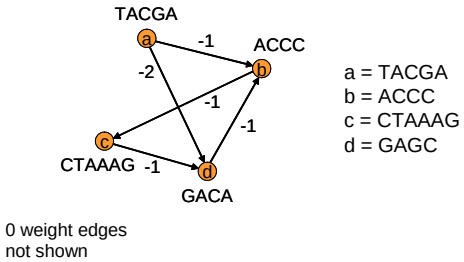


Figure 3: Example of overlap graph

Every simple path P in the overlap graph involving a set of vertices (fragments) A defines a superstring $s(P)$ for the set A, where $s(P)$ is the string obtained by concatenating all the fragments in A in order of their appearance on P while using the maximum overlap between any two adjacent fragments.
The following example illustrates the idea:

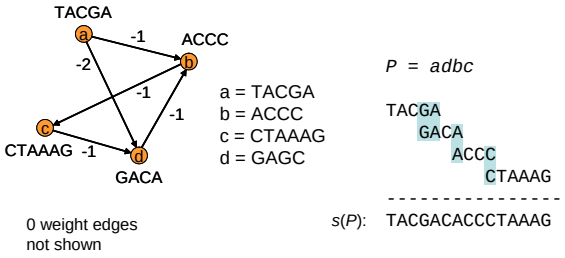


Figure 4: Superstring $s(P)$ for a path P

Therefore, a Hamiltonian path in the overlap graph defines a superstring for the set of fragments F. Note that a Hamiltonian path must exist because the graph is complete (in fact we have many, how many?).

Is the converse true? i.e. does a superstring define a Hamiltonian path in the overlap graph? The answer to this question is obviously no since a superstring can contain arbitrary characters that are not present in any fragment. But a shortest supersting cannot waste any characters. So does a shortest superstring define Hamiltonian path in the overlap graph. The answer is affirmative if F is substring-free, i.e. no fragment in F is a (strict) substring of another. First we show an example where F is not substring-free and the shortest superstring does not define a Hamiltonian path in the overlap graph.

In the example of Figure 5, the shortest covering string is $AGCT$. However, there is no Hamiltonian path P such that $s(P) = AGCT$.

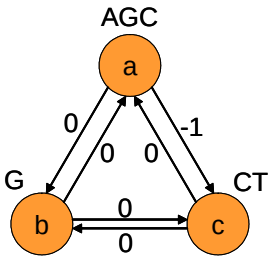
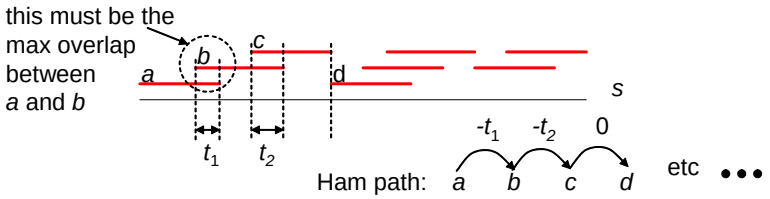


Figure 5: Shortest covering string does not define a Hamiltonian path

Let F be a substring-free set, the for every shortest superstring s, there is a Hamiltonian path P in the overlap graph G of F, such that $s(P) = s$.

Proof: The fragments appear in the shortest superstring s in some order. This order is the same order for their left end points and for their right end points in s since no fragment can be contained in another. Moreover, there can be no gaps in s between fragments (otherwise s would not be shortest). Without loss of generality this is illustrated in the figure below:



Since every two adjacent fragments must share the maximum overlap in the shortest superstring, the Hamiltonian path can be extracted from the shortest superstring as shown above.

The assumption on F begin a substring-free set is not a very constraining one since any set of fragments can be filtered and all fragments in F that are substrings of other fragments in F can be removed to obtained a new substring-free set F'. The important observation here is that removing these fragments causes no harm since a superstring for F' is a superstring for F and a superstring for F is a superstring for F'. Therefore, we can always work with a substring-free set.

Let P be a Hamiltonian path and $w(P)$ be the weight of path P (i.e. sum of weights of the edges on that path). Let $m = \sum_{f \in F} |f|$. Then

$$|s(P)| = m + w(P)$$

The proof of the above equality is simple, for instance refer to Figure 4 to verify it.

Now we establish a way to determine the shortest superstring for the set of fragments F by choosing the minimum weight Hamiltonian path in the overlap graph of F.

Let P be a Hamiltonian path with minimum weight in the overlap graph of F, then $s(P)$ is a shortest superstring for F.

Proof: Let s be a shortest superstring with $|s| < |s(P)|$. Then there is a Hamiltonian path P' such that $s = s(P')$. Now $m + w(P') = |s(P')| = |s| < |s(P)| = m + w(P)$ Therefore, $w(P') < w(P)$, a contradiction.

The Hamiltonian path approach to solving the shortest superstring is not an efficient one because finding the minimum weight Hamiltonian path is NP-hard (you can reduce HAMPATH to it). Unfortunately, there is no "better" approach to solving the shortest superstring problem because it itself is an NP-hard problem. Below we consider a greedy algorithm for finding a Hamiltonian path (not guarantted to be of minimum weight).

The greedy algorithm start with an empty path and repeatedly adds to it the least weighted available edge until it becomes a Hamiltonian path.

Note that for the path to remain a valid path, every time we add an edge (u, v), we need to check for three things:

- (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

The cycle check can be done efficeintly in $O(\log n)$ time by using a disjoint set data structure. The other two checks can be done in constant time by keeping track of the incoming and outgoing degree of each vertex.

Greedy Algorithm

sort edges by their weight: e_1, e_2, ..., $e_{|E|}$
for all $v \in V$
    $in(v) \leftarrow 0$
    $out(v) \leftarrow 0$
$H \leftarrow \emptyset$
$i \leftarrow 1$
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
                then $H \leftarrow H \cup e_i$
                    $out(u) \leftarrow 1$
                    $in(v) \leftarrow 1$
                    $i \leftarrow i + 1$

The cycle check can be implemented using a disjoint sets data structure as follows: Initially every vertex is in a distinct set. When an edge (u, v) is processed, (u, v) does not create a cycle iff u and v are in separate sets. Once edge (u, v) is added to the Hamiltonian path H, the two sets of u and v are joined together. Every disjoint sets operation (checking whether u and v are in the same set, or joining two sets) can be done in $O(\log n)$ time where n is the number of vertices (fragments).

Therefore, the running time of the above algorithm is $O(n^2 \log n)$ since we have to first sort the $O(n^2)$ edges and then process possibly $O(n^2)$ edges where each processing step requires $O(\log n)$ time dominated by the disjoint set data structure.

However, the running time of $O(n^2 \log n)$ assumes that we already have the overlap graph, which is not true. The overlap graph can be trivially constructed in $O(m^2)$ by determining the overlap of every pair of fragments. Determining the overlap of fragments a and b takes $O(|a||b|)$ time. Determining the overlap of fragment a with all other fragments therefore takes $O(|a|m)$. Doing this for all fragments in F yields an $O(m^2)$ time.

Determining the overlaps can be also done in $O(mn)$ time using a suffix tree data structure. An optimal $O(n^2 + m)$ time bound (why is that optimal?) can be also obtained using suffix trees. (this is known as the all-pairs suffix prefix problem). We will study suffix trees later.

Therefore, the greedy algorithm can be perfomed in $O(n^2 \log n + m)$.

Sequencing by hybridization

Sequencing by hybridization is another technique used for DNA sequencing. Hybridization data is obtained for the DNA with **all** possible probes of length l (for instance 4^l probes). If there are no hybridization errors, the hybridization data identifies all substrings of length l contained in the DNA sequence.

As before, the goal is to reconstruct the DNA from those substrings. The shortest superstring can be solved using a Hamiltonian approach as as before; however, we can simplify the model a little bit.

Sequencing by hybridization is a special case of the shortest superstring problem where all fragments in F has the same length l. In the overlap graph, we can keep only the edges with weights equal to $l - 1$. By construction of the fragments in F (F is the set of all substrings of length l), we know that there must be a Hamiltonian path in this modified overlap graph. Moreover, all the Hamiltonian paths now have the same weight of $-(l - 1)(n - 1)$.

Therefore, the problem reduces to finding a Hamiltonian path (weight is not important). This is still an NP-complete probelm; however, the absence of weights motivates another way to model the problem that will lead to a polynomial time algorithm.

Here's the idea: Instead of representing fragments as vertices, we will represent them as edges. Then instead of looking for a Hamiltonian path (a path that goes through every vertex once), we will look for an Euler path (a path that goes through every edge once). An Euler path can be found in linear time.

So we will construct the following graph $G = (V, E)$:

- V: $(l-1)$ length fragments (these can be obtained from our set F by considering the first and last $l-1$ characters of each fragment)
- E: a directed edge (u, v) for each fragment in F that starts with u and ends with v

Here's an example:

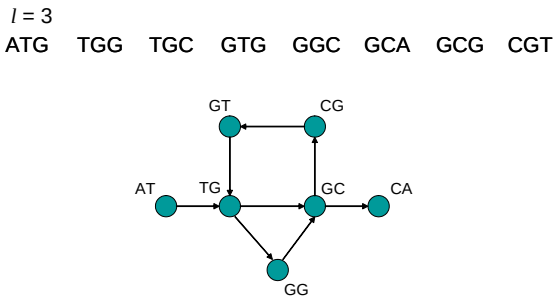


Figure 6: Fragments as edges

By construction of the fragments, we know that the graph will have all vertices balanced except possibly for two unbalanced vertices (each occurrence of an $l-1$ fragment is shared by two l length fragments, except possibly for the first and last one). By adding an edge between two unbalanced vertices we can make the graph balanced. Then we can find an Euler cycle in the graph in linear time (since the graph is balanced, there is one). To construct an Euler cycle, start fromany arbitrary edge in G and form a "random" trail by extending the already existing trail with arbitrary new edges. The procedure ends when all edges incident to a vertex in G are used in the trail. Since every vertex in G is balanced, every such trail starting at vertex v will end at v. If the trail traversed all edges, we are done. If not, then it must contain a vertex w that still has a number of untraversed edges. Note that all vertices in the graph of untraversed edges are balanced and, therefore, there exists a random trail starting at w and containing only untraversed edges. Once can now enlarge the random trail as follows: insert a random trail of untraversed edges from w at some point in the random trail from v where w is reached. Repeating this will eventually yield an Euler cycle. This algorithm can be implemented in linear time. The Euler cycle will give an Euler path.

References

Setubal J., Meidanis J., Introduction to Computational Molecular Biology, Chapter 4.
Pevzner P., Computational Molecular Biology, Chapter 5.