CSC2427 (Winter 2006)
Algorithms in Molecular Biology (Prof. M. Brudno)
Lecture Notes — January 18, 2006
Part I: Sequence Alignment

Scribe: Ulrich Germann

1 Introduction

Current DNA sequencing methods can accurately determine only a few hundred 'letters' (bases) off each end of a contiguous stretch of DNA.[1] It is therefore impossible to read a complete genome all at once. The human genome, for example, consists of about 3 billion base pairs — in contrast to about 500 that can be read at a time. The *Whole Genome Shotgun Sequencing Method* addresses this problem by randomly cutting multiple samples of identical DNA into small chunks, each of which is sequenced separately.

If we were to cut up only a single DNA sample into many pieces, there would be no way of reconstructing the original sequence from the individual pieces. By cutting up multiple copies of the sequence in different ways, we can reconstruct the original sequence by aligning segments with overlapping reads, as illustrated in Fig. 1.

Reads are asymmetric: They have an 'outside' end (where the DNA has been cut) and an 'inside' end (where the chunk of DNA continues but can't be read).[2] Reads can overlap in three ways ('$\cdots$' indicates the inside end of a read):

1. The outside end of one read overlaps with the inside end of the other:

[1] More precisely, DNA sequencing always proceeds from the 5′ ("five prime") end of a single strain of DNA a to its 3′ end. Since genome information is chemically encoded in two complementary strands of DNA that are arranged 'head to toe' in the double helix, we can simplify our language by assuming a single sequence of letters (bases) that is read from both ends. Reading a sequence in reverse order technically means forming the complement (substitute A for T and vice versa, and C for G and vice versa) and reading that complement in reverse order. The limitation on the number of letters that can be read at a time results from the decreasing relative distance between adjacent positions as the DNA chain gets longer: $\frac{501}{500} \ll \frac{51}{50}$ (comment by Prof. Steipe in class; see also Wikipedia:DNA_sequencing).

[2] Another way to look at it is to say that since we always read from the outside in, but we don't know which end we are reading from, we have to consider each sequence of letters in both read directions, from left to right and from right to left.

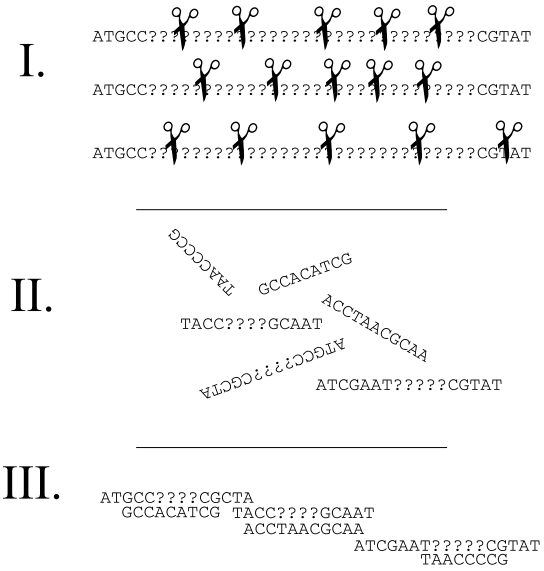
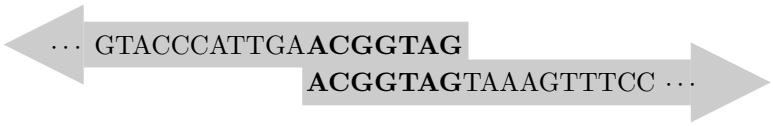
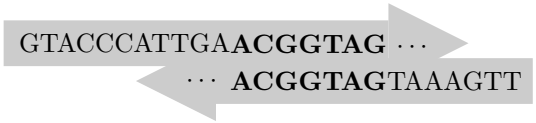


Figure 1: The *Whole Genome Shotgun Method*: Multiple identical DNA samples are randomly cut into small pieces (I), which are sequenced individually, resulting in a set of reads (II). By looking for overlaps between reads, we can reconstruct the originial DNA sequence (III).

2. The two 'outside' ends of the reads overlap.



3. The two 'inside' ends of the reads overlap:



2 Sequence assembly as a Hamiltonian Path Problem

2.1 The Shortest Common Superstring Problem

DNA sequence assembly can be expressed as a variant of the problem of finding the *shortest common superstring* (SCS) for a set of strings (Kececioglu & Myers, 1995). By casting the problem this way, we assume that the shortest sequence hypothesis is the correct one, which is not guaranteed but generally works well as a rule of thumb in practice. Given a set $\mathcal{S}$ of strings (the reads), we try to find the shortest string that contains all the strings in $\mathcal{S}$. For example given the set $\mathcal{S} = \{$ `AABBA, BABAA, BAAAA, AAABA, ABBAB` $\}$, the SCS is `AABBABAAAABA` with 12 letters (in contrast to 25 letters if we simply concatenate all strings). We can reduce the SCS problem to a Hamiltonian path problem in a directed graph as follows.

Each string s_i in $\mathcal{S}$ is represented by a vertex v_i. Each vertex is connected with all the other vertices by weighted, directed edges e_{ij}, whose weight w_{ij} is given by the formula $w_ij = \mathit{length}(s_i) + \mathit{length}(s_j) - \mathit{length}(\mathit{overlap}(s_i, s_j))$. For example, the cost of going from the vertex representing `BABAA` to the vertex representing `BAAAA` is $5 + 5 - 3 = 7$ — the shortest string that contains both `BABAA` and `BAAAA` is 7 characters long (`BABAAAA`). The entire search space looks

like this:

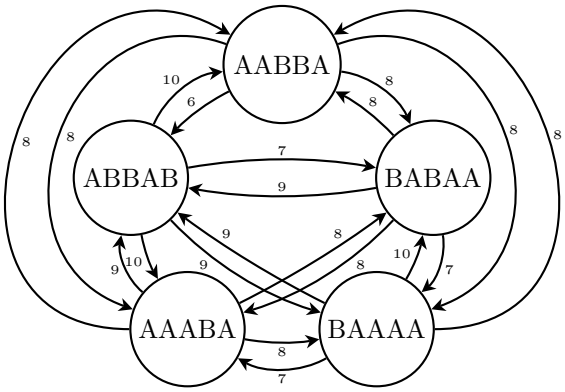


Figure 2: Search graph for the *Shortest common superstring* problem. Edge weights are given by the length of simple concatenation minus the number of letters saved by overlap. The shortest path through the graph that visits each vertex exactly once (Hamiltonian Path) represents the shortest common superstring.

The walk through the graph that visits every node at least once while accumulating the smallest number of points along the edges corresponds to the shortest superstring of the set of strings.

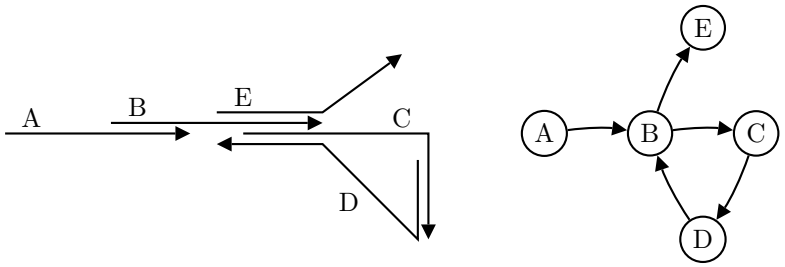
2.2 Necessary adaptations for sequence assembly

For DNA sequence assembly, we have to make a few modifications. The first is that we allow edges only between vertices denoting segments with sufficient overlap. Small overlaps should be ignored, because there's a good chance that they are accidental — with a vocabulary of only four letters, there's at least[3] a 25% chance of accidental overlap of one letter, a chance of $\frac{1}{16}$ for an overlap of two letters, and so forth. At the same time, we want to allow small differences between overlapping segments, because the methods with which we obtain those reads are not perfect and sequencing errors do occur. With some knowledge of how often these errors occur, we can develop probabilistic models that determine the probability that two sequences overlap. For the remainder of this presentation we assume that all sequencing errors have been accounted for and corrected, so that we are dealing with error-free reads.

Secondly, because real-life DNA contains repetitions of often considerable length, we cannot require that each node be visited *exactly* once. A valid solution may require us to pass through a node multiple times.

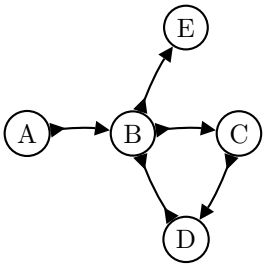
Thirdly, we have to represent the problem as a *bidirected* graph instead of a directed one. The reason is that we don't know the spatial orientation of the reads. They may represent a stretch of DNA in order (left–to–right), or in reverse order (right–to–left), leading to the three forms of overlaps mentioned earlier. Unlike the case of the shortest common superstring, the overlap relation for reads is symmetrical. If A overlaps with B, then B overlaps with A when both are read in reverse order. It might be tempting to use directed edges to indicate read direction, but this does not work (nor do undirected edges). Imagine the following situation involving five reads A, B, C, D, E: A overlaps with B, B with C and E, and C with D (all in the same read direction only), and D overlaps with the reverse (only) of B, as shown below on the left. If we use directed edges to indicate read direction, the corresponding graph looks like like the one on the right.

[3]This is the case if letter occurrences are uniformly distributed.



This graph correctly allows the sequence *A*–*B*–*C*–*D*–*B*. This sequence is valid because the reverse of *B* may indeed occur independently of the *B* somewhere in the underlying DNA sequence. Unfortunately, the graph fails to rule out the sequence *A*–*B*–*C*–*D*–*B*–*E*, which is invalid because it uses the same end of *B* to connect *D* and *E*, which would be equivalent to a fork in the DNA sequence.

Bidirected graphs address this dilemma by enforcing directionality *through* instead of *between* nodes. They are easiest to understand by imagining that each node has two ends, say, left (L) and right (R). Edges can be traveled in either direction but always connect two specific ends. Thus, we have three types of edges: L–L, R–R, and L–R/R–L. Valid paths are only those that enter each node on one end and leave it on the other. If we use outward–pointing arrows ($\rightarrow$ / $\leftarrow$) to indicate connection at the left end of the node and inward–pointing arrows ($\dashv$ / $\vdash$) for attachment to the right end, we obtain the following graph.

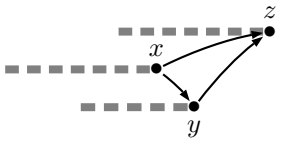


Since the path from *D* through *B* enters *B* on a $\dashv$ connection, it has to leave *B* on a $\rightarrow$ connection, which rules out the sequence *A*–*B*–*C*–*D*–*B*–*E*.

2.3 Graph simplifications

Since the Hamiltonian path problem is NP-complete, it is highly desirable to simplify the graph as much as possible. Myers (1995) lists three ways of simplifying overlap graphs:

1. Removal of vertices representing reads that are fully contained in other reads.
2. **Transitive edge reduction.** Consider the following situation.



The path $x \rightarrow y \rightarrow z$ and the edge $x \rightarrow z$ encode exactly the same information. We can therefore remove the edge $x \rightarrow z$ without losing the ability to represent the underlying overlap.

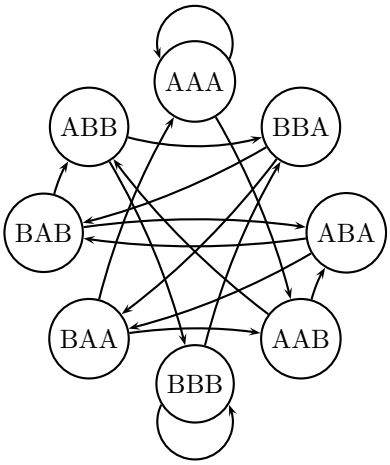


Figure 3: A 3-dimensional de Bruijn graph over the vocabulary $\{A, B\}$.

3. **Collapsing vertex chains.** A vertex chain is a sequence of vertices with 'in' and 'out' degree of 1. $\rightarrow x \rightarrow y \rightarrow z \rightarrow$. We can replace such chains by single edges labeled with the concatenation of the individual edge labels. Since the original vertex chain was the only way to reach the 'internal' vertices, we also know that this new edge must be contained in the final solution at least once.

3 Sequence assembly as an Eulerian Path Problem

Pevzner *et al.* (2001) cast the problem of sequence assembly as that of finding an *Eulerian Superpath* through a k-dimensional *de Bruijn graph* (Fig. 3). A k–dimensional de Bruijn Graph over a vocabulary V consists of a set of $|V|^k$ vertices representing distinct k–tuples over V, and directed edges between them representing $(k+1)$–tuples. That is, two vertices are connected by a directed graph if the last $k-1$ elements of the tuple represented by the first vertex are identical to the first $k-1$ elements of the second vertex.

An Eulerian path of a graph is a path (sequence of adjacent edges) that contains every edge exactly once. Such a path does not exist for all graphs. However, any connected graph can be transformed into an Eulerian graph (i.e., a graph for which an Eulerian path exits) by allowing multiple parallel edges wherever an edge is visited multiple times by a respective Chinese Postman Path (CPP). A CPP through a graph is a minimal path that visits each edge *at least* once.

A *superpath* for a set of paths $\mathcal{P}$ is a path that contains all elements of $\mathcal{P}$ as subpaths. An Eulerian superpath is a superpath that contains each edge exactly once.

Pevzner *et al.*'s sequence alignment method proceeds as follows. First, a de Bruijn-like graph $\mathcal{G}_0$ is constructed that contains an edge for each $(k+1)$–tuple that occurs at least once in any of the reads. Every read in the original data set then corresponds to a path $p_i \in \mathcal{P}_0$ through $\mathcal{G}_0$. $\mathcal{P}$ is the set of all paths corresponding observed reads. An Eulerian superpath for $(\mathcal{G}_0, \mathcal{P}_0)$ gives us the best hypothesis of the original DNA sequence. (To be precise, what we are looking for is a Chinese Postman superpath, because some edges may have to be used multiple times to cover all edges in the graph.)

To simplify the problem, $\mathcal{G}_0$ and $\mathcal{P}_0$ are subject to a series of simplifying *equivalent transformations*

$$(\mathcal{G}_0, \mathcal{P}_0) \rightarrow (\mathcal{G}_1, \mathcal{P}_1) \rightarrow \cdots \rightarrow (\mathcal{G}_k, \mathcal{P}_k)$$

In this context a transformation is equivalent if there is a one-to-one mapping between the elements of $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$ and between superpaths for $(\mathcal{G}_0, \mathcal{P}_0)$ and superpaths for $(\mathcal{G}_{i+1}, \mathcal{P}_{i+1})$. Graph and path set simplification is accomplished by two means.

***x*, *y*–detachment** Let x and y be adjacent, directed edges such that x leads from vertex v_s to vertex v_m, and y from v_m and to vertex v_e. We introduce a new edge z connecting v_s with

v_e and replace all occurrences of x, y with z in all paths $\in \mathcal{P}$. If the v_m has 'in' degree 1, i.e., x is the only edge leading to it, we also replace all occurrences of initial y in all paths $\in \mathcal{P}$ starting with y. Analogously, if v_m has 'out' degree 1, i.e., y is the only edge starting at v_m, we also replace all occurrences of final x in all paths $\in \mathcal{P}$ that end with x. If x and y are not contained in any path any more, we remove them. (Note that if v_m has 'in' or 'out' degree > 1, some instances of initial y or final x may be left.) Since we reduce the number of edges in the respective paths with each detachment, this procedure converges in the ideal case to a graph where each path (representing a read) is represented by a single edge, and the Chinese Postman superpath problem has been reduced to a regular Chinese Postman path problem.

***x*-cut** In cases that cannot be resolved by the procedure described above, path–initial or –final occurrences of an edge x can be 'cut' from the paths in $\mathcal{P}$ if the respective source vertex $v_{\rightarrow x}$ has 'out' degree 1 and the destination vertex $v_{x\rightarrow}$ 'in' degree 1. In other words, x is the only edge starting at $v_{\rightarrow x}$ and the only edge ending in $v_{x\rightarrow}$. In this case, the all path–initial or –final occurrences of x are removed from the paths in $\mathcal{P}$.

There are no guarantees that this approach actually works under all circumstances, although it seems to work in practice. Two questions in particular remain open.

1. What about read direction?
2. Is the detour via the de Bruijn graph with subsequent simplification really necessary? Can't we build the final graph directly?

As for the first issue, the description in Pevzner *et al.* (2001:9751) mentions it only in passing: The authors simply assume that the set of reads "S contains a complement of every read and that the de Bruijn graph can be partitioned into two subgraphs (the "canonical" one and its reverse complement)" but do not provide any procedure for accomplishing this subdivision.

As for second question, *string graphs* (Myers, 2005) provide a more direct implementation of the central idea of the Eulerian approach, which is to represent reads as paths in the graphs, not vertices.

4 String graphs

String graphs are constructed as follows. First, all reads that are completely contained in other reads (in practice about 40%) are filtered out. They are ignored in graph construction because they do not represent any information that is not contained in the longer reads that contain them.

For each of the remaining reads r_i, two vertices $v_{r_i.B}$ and $v_{r_i.E}$ are established in the graph, corresponding to the two ends of the read. Which end corresponds to which vertex is an arbitrary decision. Each overlap between reads is represented by two labeled, directed edges. These edges reach from the overlap–adjacent end of each read to the 'far' end of the respective other read and are labeled with the non-overlapping part of the underlying read. Figure 4 illustrates the procedure.

Every path in the resulting graph now represents a partial DNA sequence hypothesis that is consistent with the observed reads. Any path that visits at least one of the two nodes corresponding to each read at least once constitutes a valid DNA sequence hypothesis. Every hypothesis is encoded twice: once in forward direction, and once in the reverse direction.

4.1 Graph simplification

We now simplify the graph by *transitive edge reduction* and by collapsing chains of vertices with 'in' and 'out' degree 1 into single edges, as described earlier in Sec. 2.3.

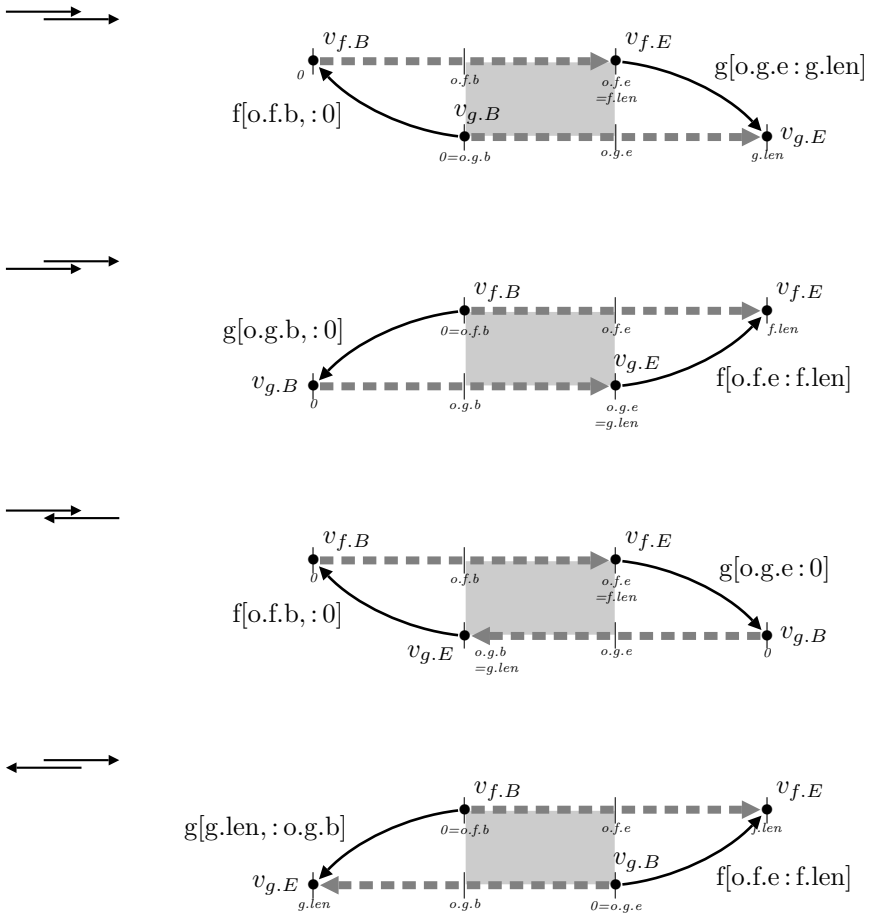


Figure 4: **String graph construction.** Each read f is represented by a vertex pair $(v_{f.B}, v_{f.E})$. The index notation $f[i : k]$ indicates substrings running from index position i up to (or down to, if $k < i$) and including k. $f.len$ is the length of the read f. $o.f.b$ marks the position where the overlap starts in f, $o.f.e$ its end. For each overlap, two edges are added, going from the overlap–adjacent end of each read to the 'far' end of the respective other. There figure shows how edges are added for each of the four ways in which reads can overlap; the first two scenarios are equivalent. Dashed arrows show read direction, and solid arrows represent directed edges. The overlapping region of the reads is represented by the gray area.

4.2 Recasting into a Hamiltonian Path Problem

We cannot use the resulting graph directly for sequence assembly. Each read is represented by two vertices, at least one of which must be visited at least once by the resulting path. We can transform the problem into a Hamiltonian problem by collapsing the two vertices corresponding to the beginning and the end of each read into one and using bidirected edges (as described in Sec. 2.2) to keep track of which vertex the edge lead to or originated from in the original graph. The advantage of Myers's method graph construction over the original formulation of the sequence alignment problem as a Hamiltonian path problem is that the resulting graphs are much simpler and leaner ("two to three orders of magnitude fewer vertices and edges", according to Myers, 2005), and therefore computationally much less expensive to search.

4.3 Sequence alignment as a network flow problem

We said earlier that if an edge is the result of collapsing a vertex chain into a single edge, we know that it must be use at least once in the final solution. We can also establish probabilistic upper bounds on the number of times we expect to traverse each edge. For this purpose, we model the 'arrival' of reads as a Poisson process. In other words, we determine the average density (arrival rate) or reads per DNA segment length. Since we don't know the length of the genome, we use bootstrapping procedure to estimate this quantity (for details see Sec. 4 in Myers, 2005). Given an edge with a certain length Δ and the number of reads n that are contained in the edge label, we can compare the probability having as many reads as we have observed under the hypothesis that the underlying sequence occurs only once with the probability of our observation under the hypothesis that the underlying sequence is a repeat. Suppose, for example, that we encounter a new read every 200 base pairs. Then, if we have 3 reads matching an edge of label length 1000, the respective sequence is most likely unique. If we observe, on the other hand, 10 matching reads, the respective sequence most likely occurs more than once. Myers (2005) distinguishes only singletons from multiply occurring sequences but does not estimate how often individual repeats occur.

The sequence alignment problem can then be cast as a generalized Eulerian tour problem (visit every edge at least $m \geq 0$ and at most $n \geq 1$ times, where m and n are edge-specific) and solved via Integer Programming techniques, in particular network flow analysis.

References

Kececioglu, J. D. & E. W. Myers. 1995. "Combinatorial algorithms for dna sequence assembly." *Algorithmica*, 13:7–51.

Myers, Eugene W. 1995. "Toward simplifying and accurately formulating fragment assembly." *Journal of Computational Biology*, 2(2):275–290.

Myers, Eugene W. 2005. "The fragment assembly string graph." *Bioinformatics*, 21(2):ii79–ii85.

Pevzner, Pavel A., Haixu Tang, & Michael S. Waterman. 2001. "An eulerian path approach to dna fragment assembly." *Proceedings of the National Academy of Sciences of the United States of America*, 98(17):9748–9753.