A New Approach to Fragment Assembly in DNA Sequencing

Pavel A. Pevzner
Department of Computer Science and Engineering
University of California at San Diego
La Jolla, CA, USA
ppevzner@cs.ucsd.edu

Haixu Tang
Department of Mathematics
University of Southern California
Los Angeles, CA, USA
tanghx@hto.usc.edu

Michael S. Waterman
Department of Mathematics
University of Southern California
Los Angeles, CA, USA
msw@hto.usc.edu

ABSTRACT

For the last twenty years fragment assembly in DNA sequencing followed the "overlap - layout - consensus" paradigm that is used in all currently available assembly tools. Although this approach proved to be useful in assembling clones, it faces difficulties in genomic shotgun assembly: the existing algorithms make assembly errors and are often unable to resolve repeats even in prokaryotic genomes. Biologists are well-aware of these errors and are forced to carry additional experiments to verify the assembled contigs.

We abandon the classical "overlap - layout - consensus" approach in favor of a new Eulerian Superpath approach that, for the first time, resolves the problem of repeats in fragment assembly. Our main result is the reduction of the fragment assembly to a variation of the classical Eulerian path problem. This reduction opens new possibilities for repeat resolution and allows one to generate error-free solutions of the large-scale fragment assembly problems. The major improvement of EULER over other algorithms is that it resolves all repeats except long perfect repeats that are theoretically impossible to resolve without additional experiments.

1. INTRODUCTION

For the last twenty years fragment assembly in DNA sequencing mainly followed the "overlap - layout - consensus" paradigm, that is used in all currently available software tools for fragment assembly [?, 2, 18, 7]. Although this approach proved to be useful in assembling contigs of moderate sizes, it faces difficulties while assembling prokaryotic genomes a few million bases long. These difficulties led to introduction of the double-barreled DNA sequencing [17, 21] that uses additional experimental information for assembling large genomes in the framework of the same "overlap - layout - consensus" paradigm [11].

Although the classical approach culminated in some excellent fragment assembly tools (Phrap, CAP3, TIGR, and Celera assemblers are among them), critical analysis of the "overlap - layout - consensus" paradigm reveals some weak points. First, the overlap stage finds *pairwise* similarities that do not always provide true information on whether the fragments (sequencing reads) overlap. A better approach would be to reveal *multiple* similarities between fragments since sequencing errors tend to occur at random positions while the differences between repeats are always at the same positions. However, this approach is infeasible due to high computational complexity of the multiple alignment problem. Another problem with the conventional approach to fragment assembly is that finding the correct path in the overlap graph with many false edges (layout problem) becomes very difficult.

Unfortunately, these problems are difficult to overcome in the framework of the "overlap - layout - consensus" approach and the existing fragment assembly algorithms are often unable to resolve the repeats even in prokaryotic genomes. Inability to resolve repeats and to figure out the order of contigs leads to additional experimental work to complete the assembly [19]. Moreover all the programs we tested made errors while assembling shotgun reads from the bacterial sequencing projects *Campylobacter jejuni* [13], *Neisseria meningitidis* [12], and *Lactococcus lactis* [1]. Biologists at large sequencing centers are well-aware of potential assembly errors and are forced to carry additional experimental tests to verify the assembled contigs. Bioinformaticians are also aware of assembly errors as evidenced by finishing software that supports experiments correcting these errors [4].

How can one resolve these problems? Surprisingly enough, an unrelated area of DNA arrays provides a hint. *Sequencing by Hybridization (SBH)* is a 10-years old idea that never became practical but (indirectly) created the DNA arrays industry. Conceptually, SBH is similar to fragment assembly, the only difference is that the "reads" in SBH are much shorter l-tuples. In fact, the very first attempts to solve the SBH fragment assembly problem [3, 9] followed the "overlap-layout-consensus" paradigm. However, even in a simple case of error-free SBH data, the corresponding layout problem leads to the NP-complete *Hamiltonian Path Problem*. Pevzner, 1989 [15] proposed a different approach that reduces SBH to an easy-to-solve *Eulerian Path Problem* in the *de Bruijn* graph by abandoning the "overlap-layout-consensus" paradigm.

Since the Eulerian path approach transforms a once difficult layout problem into a simple one, a natural question is: "Could the Eulerian path approach be applied to fragment assembly?". Idury and Waterman, 1995 answered this question by mimicking the fragment assembly problem as an SBH problem [8]. They represented every read of length n as a collection of $n - l + 1$ l-mers and applied an Eulerian path algorithm to a set of l-tuples formed by the union of such collections for all reads. At the first glance this transformation of every read into a collection of l-tuples is a very short-sighted procedure since information about the sequencing reads is lost. However, the loss of information is minimal for large l and is well paid for by the computational advantages of the Eulerian path approach in the resulting easy-to-analyze graph. Not to mention that the lost information can be easily restored at the later stages.

Unfortunately, the Idury-Waterman approach, while very promising, did not scale up well. The problem is that the sequencing errors transform a simple de Bruijn graph (corresponding to an error-free SBH) into a tangle of erroneous edges. For a typical sequencing project, the number of erroneous edges is a few times larger than the number of real edges and finding the correct path in this graph is extremely difficult, if not impossible task. Moreover, repeats in prokaryotic genomes pose serious challenges even in the case of error-free data since the de Bruijn graph gets very tangled and difficult to analyze.

This paper abandons the classical "overlap-layout-consensus" approach in favor of a new Eulerian superpath approach. Our main result is the reduction of the fragment assembly problem to a variation of the classical Eulerian path problem. This reduction opens new possibilities for repeat resolution and leads to the EULER software that generated optimal solutions for the large-scale assembly projects that were studied.

2. NEW IDEAS

Given two similar reads, how can we decide whether they correspond to the same region (i.e. the differences between them are due to sequencing errors) or to two copies of a repeat located in different parts of the genome? This problem is crucial for all fragment assembly algorithms and pairwise comparison used in the conventional algorithms does not adequately resolve this problem. Our error-correction procedure implicitly uses multiple comparison of reads and successfully distinguishes these two situations.

Both Idury and Waterman, 1995 [8] and Myers,1995 [10] tried to deal with errors and repeats via graph reductions. In fact, there are some conceptual similarities between [8] and [10] (although the corresponding graphs are very different). However, both these methods do not explore multiple alignment of reads to *fix* sequencing errors at the *pre-processing* stage. Of course, multiple alignment of reads is costly and pairwise alignment is the only realistic option at the overlap stage of the conventional fragment assembly algorithms. However, the multiple alignment becomes feasible when we deal with perfect or nearly perfect matches of short l-tuples, exactly the case in the SBH approach to fragment assembly. Our *error correction* idea utilizes the multiple alignment of short substrings to modify the original reads and to create a new instance of the fragment assembly problem with the greatly reduced number of errors. The error correction makes our reads almost error-free and transforms the original very large graph into a graph with very few erroneous edges. In some sense, the error correction is a variation of the consensus step taken at the very first step of fragment assembly (rather than at the last one as in the conventional approach).

Imagine an ideal situation when the error-correction procedure eliminated all errors and we deal with a collection of error-free reads. Is there an algorithm to reliably assemble such error-free reads in a large-scale sequencing project? At the first glance, the problem looks simple, but surprisingly enough, the answer is no: we are unaware of any algorithm that solves this problem. For example, Phrap, CAP3 and TIGR assemblers make 17, 14, and 9 assembly errors correspondingly while assembling *real* reads from the *N. meningitidis* genome. All these algorithms still make errors while assembling the *error-free* reads from the *N. meningitidis* genome (although the number of errors reduces to 5, 4, and 2 correspondingly). Although the TIGR assembler makes less errors than other programs, this accuracy does not come for free, since this program produces twice as many contigs as do the other programs. EULER made no assembly errors and produced less contigs with *real data* than other programs produced with *error-free* data! EULER can be also used to immediately improve the accuracy of Phrap, CAP3 and TIGR assemblers: these programs produce better assemblies if they use error-corrected reads from EULER.

To achieve such accuracy, EULER has to overcome the bottleneck of the Idury-Waterman approach and to restore information about sequencing reads that was lost in the construction of the de Bruijn graph. Our second *Eulerian Superpath* idea addresses this problem. Every sequencing read corresponds to a path in the de Bruijn graph called a *read-path*. An attempt to take into account the information about the sequencing reads leads to the problem of finding an Eulerian path that is consistent with all read-paths, an Eulerian Superpath Problem. Below we show how to solve this problem.

This simple description hides some algorithmic challenges that will be discussed later.

3. ERROR CORRECTION

Sequencing errors make implementation of the SBH-style approach to fragment assembly difficult. To bypass this problem we reduce the error rate by a factor of 35-50 at the pre-processing stage and make the data almost error-free by solving the Error Correction Problem. We use the *N. meningitidis* (NM) sequencing project completed at the Sanger Center [12] as an example. NM is one of the most "difficult-to-assemble" bacterial genome completed so far. It has 126 long perfect repeats up to 3832 bp in length (not to mention many imperfect repeats). The length of the genome is 2,184,406 bp. The sequencing project resulted in 53263 reads of average length 400 (average coverage is 9.7). There were 255,631 errors overall distributed over these reads. It results in 4.8 errors per read (error rate of 1.2%).

Let s be a sequencing read (with errors) derived from a genome G. If the sequence of G is known then the error correction in s can be done by aligning the read s against the genome G. In real life, the sequence of G is not known until the very last "consensus" stage of the fragment assembly. It is a catch-22: to assemble a genome it is highly desirable to correct errors in reads first, but to correct errors in reads one has to assemble the genome first. To bypass this catch-22, let's assume that, although the sequence of G is unknown, the set G_l of all continuous strings of fixed length l (l-tuples) present in G is known. Of course, G_l is unknown either, but G_l can be reliably *approximated without knowing the sequence of* G. An l-tuple is called *solid* if it belongs to more than M reads (where M is a threshold) and *weak* otherwise. A natural approximation for G_l is the set of all solid l-tuples from a sequencing project.

Let T be a collection of l-tuples called a *spectrum.* A string s is called a T*-string* if all its l-tuples belong to T. Our approach to error correction leads to the following

Spectral Alignment Problem. Given a string s and a spectrum T, find the minimum number of mutations in s that transform s into a T-string.

A similar problem was considered by Peer and Shamir, 2000 [14], in a different context of resequencing by hybridization. In the context of error corrections, the solution of the Spectral Alignment Problem makes sense only if the number of mutations is small. In this case the Spectral Alignment Problem can be efficiently solved by dynamic programming even for large l (compare with [14]).

Spectral alignment of a read against the set of all solid l-tuples from a sequencing project, suggests the error corrections that may change the sets of weak and solid l-tuples. Iterative spectral alignments with the set of all reads and all solid l-tuples gradually reduce the number of weak l-tuples, increase the number of solid l-tuples, and reads and all solid l-tuples gradually reduce the number of weak l-tuples, increase the number of solid l-tuples, and lead to elimination of many errors in bacterial sequencing projects. Although the Spectral Alignment Problem helps to eliminate errors (and we use it as one of the steps in EULER) it does not adequately capture the specifics of the fragment assembly. The Error Correction Problem described below is somewhat less natural than the Spectrum Alignment Problem but it is probably a better model for fragment assembly (although it is not a perfect model either). The greedy heuristics for the Error Correction Problem eliminates up to $\approx 97\%$ of errors in a typical bacterial project.

Given a collection of reads (strings) $S = \{s_1, \ldots, s_n\}$ from a sequencing project and an integer l, the spectrum of S is a set S_l of all l-tuples from the reads $s_1, \ldots, s_n$ and $\overline{s}_1, \ldots, \overline{s}_n$, where $\overline{s}$ denotes a reverse complement of read s. Let Δ be an upper bound on the number of errors in each DNA read. A more adequate approach to error correction motivates the

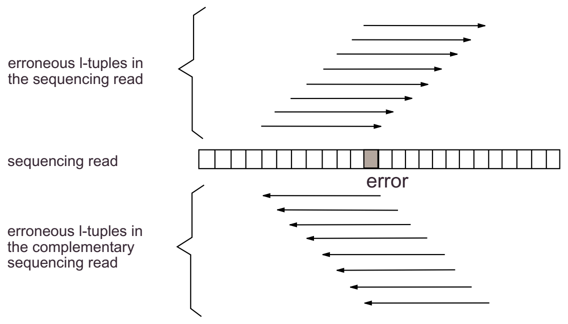
erroneous l-tuples in the sequencing read
sequencing read
error
erroneous l-tuples in the complementary sequencing read

Figure 1: An error in a read affects l l-tuples in this read and l l-tuples in the complementary read, creating $2l$ erroneous l-tuples.

following

Error Correction Problem. Given S, Δ, and l, introduce up to Δ corrections in each read in S in such a way that $|S_l|$ is minimized.

An error in a read s affects at most l l-tuples in s and l l-tuples in $\overline{s}$ (Fig. 1) and usually creates $2l$ erroneous l-tuples that point out to the same sequencing error ($2d$ for positions within a distance $d < l$ from the endpoint of the reads). Therefore a greedy approach for the Error Correction Problem is to look for an error correction in the read s that reduces the size of S_l by $2l$ (or $2d$ for positions close to the endpoints of the reads). This simple procedure already eliminates 86.5% of errors in sequencing reads. Below we describe a more involved approach that eliminates 97.7% of sequencing errors. This approach transforms the original fragment assembly problem with 4.8 errors per read on average into an almost error-free problem with 0.11 errors per read on average.

Two l-tuples are called *neighbors* if they are one mutation apart. For an l-tuple a define its multiplicity $m(a)$ as the number of reads in S containing this l-tuple. An l-tuple is called an *orphan* if (i) it has small multiplicity, i.e., $m(a) \leq M$, where M is a threshold, (ii) it has the only neighbor b, and (iii) $m(b) \geq m(a)$. The position where an orphan and its neighbor differ is called an *orphan position.* A sequencing read is *orphan-free* if it contains no orphans.

An important observation is that each erroneous l-tuple created by a sequencing error usually does not appear in other reads and is usually one mutation apart from a real l-tuple (for an appropriately chosen l). Therefore, a mutation in a read usually creates $2l$ orphans. This observation leads to an approach that corrects errors in orphan positions within the sequencing reads, if the overall number of error corrections in a given read to make it orphan-free is at most Δ. The greedy *orphan elimination* approach to the Error Correction Problem starts error corrections from the orphan positions that reduce the size of S_l by $2l$ (or $2d$ for positions at distance $d < l$ from the endpoints of the reads). After correcting all such errors the "$2l$ condition" gradually transforms into a weaker $2l - \delta$ condition.

4. ERROR CORRECTION OR DATA CORRUPTION?

A word of caution is in place. Our error-correction procedure is not perfect while deciding which nucleotide, among, let's say, A or T is correct in a given l-tuple within a read. If the correct nucleotide is A, but T is also present in some reads covering the same region, the error-correction procedure may assign T instead of A to all reads, i.e., to introduce an error, rather than to correct it (particularly, in the low-coverage regions). Since our algorithm sometimes introduces errors, *data corruption* is probably a more appropriate name for this approach! Introducing an error in a read is not such a bad thing as long as the errors from overlapping reads covering the same position are *consistent* (i.e., they corresponds to a single mutation in a genome). An important insight is that, at this stage of the algorithm, we don't care much whether we correct or introduce errors in the sequencing reads. From algorithmic perspective, introducing an error, which simply corresponds to changing a nucleotide in a final assembly, is not a big deal. It is much more important to make sure that we eliminate a competition between A and T at this stage, thus reducing the complexity of the de Bruijn graph. In this way we eliminate false edges in our graph and deal with this problem later: the correct nucleotide can be easily reconstructed at the final consensus stage of the algorithm. For *N. meningitidis* sequencing project, orphan elimination corrects 234410 errors, and introduces 1452 errors. It leads to a tenfold reduction in the number of sequencing errors (0.44 errors per read).

The orphan elimination procedure is usually run with $M = 2$ since orphan elimination with $M = 1$ leaves some errors uncorrected. For a sequencing project with coverage 10 and error rate 1%, every solid 20-tuple has on average 2 orphans o_1 and o_2, each with multiplicity 1 (i.e., an expected multiplicity of this 20-tuple is 8 rather than 10 as in the case of error-free reads). With some probability, the same errors in (different) reads correspond to the same position in the genome thus "merging" o_1 and o_2 into a single l-tuple o with $m(o) = 2$. Although the probability of such event is relatively small, the overall number of such cases is large for large genomes. In our studies of bacterial genomes setting $M = 2$ and simultaneous correction of up to M multiple errors worked well in practice. With $M = 2$, we eliminated additional 705 errors and created 131 errors (21837 errors, or 0.41 errors per read are left).

Orphan elimination is a more conservative procedure than spectral alignment. Orphans were defined as l-tuples of low multiplicity that have only one neighbor. The latter condition (that is not captured by the spectral alignment) is important since in the case of multiple neighbors it is not clear how to correct an error in an orphan. For the *N. meningitidis* genome there were 1862 weak 20-mers ($M \leq 2$) that had multiple neighbors. Our approach to this problem is to increase l in a hope that there is only one "competing" neighbor for longer l. After increasing l from 20 to 100, the number of orphans with multiple neighbors have been reduced from 1862 to 17.

Orphan elimination should be done with caution since errors in reads are sometimes hard to distinguish from differences in repeats. If we treated the differences between repeats (particularly repeats with low coverage) as errors, then orphan elimination would correct the differences between repeats instead of correcting errors. This may lead to inability to resolve repeats at the later stages.

It is important to realize that error corrections in orphan positions often create new orphans. Imagine a read containing an imperfect low-coverage ($\leq M$) copy of a repeat that differs from a high-coverage ($> M$) copy of this repeat by a substitution of a block of t consecutive nucleotides. Without knowing that we deal with a repeat, the orphan elimination procedure would first detect two orphans, one of them ending in the first position of the block, and the other one starting in the last position of the block. If the orphans are eliminated without checking the "at most Δ corrections per read" condition, these two error corrections will shrink the block to the size $t - 2$ and will create two new orphans in the beginning and the end of this shrunk block. At the next step, this procedure would correct the first and the last nucleotides in the shrunk block, and, in just $t/2$ steps, erase the differences between two copies of the repeat.

Of course, for long bacterial genomes many "bad" events that may look improbable happen and there are two types of errors that are prone to orphan elimination. They require a few coordinated error corrections since single error corrections do not lead to a significant reduction in the size of S_l and thus may be missed by the greedy orphan elimination. These errors include: (i) consecutive or closely-spaced errors in the same read and (ii) the same error with high multiplicity ($> M$) at the same genome position in different reads.

The first type of error is best addressed by solving the Spectral Alignment Problem to identify reads that require less than Δ error corrections. We found that some reads from the *N. meningitidis* project have very poor spectral alignment. These reads are likely to represent contamination, vector, isolated reads, or an error in the sequencing pipeline [4]. All these reads are of limited interest and should be discarded. In fact, it is a common practice in sequencing centers to discard such "poor-quality" reads and we adopt this approach. Although deleting poor-quality reads may slightly reduce the amount of available sequencing information, it greatly simplifies the assembly process. Another important advantage of spectral alignment is an ability to identify the chimeric reads. Such reads are characterized by good spectral alignments of the prefix and suffix parts, that, however, cannot be extended to a good spectral alignment of the entire read. EULER breaks the chimeric reads into two or more pieces and preserves the sequencing information from their prefix and suffix parts.

The second type of error reflects the situation with M identical errors in different reads corresponding to the same genome position and generating an erroneous l-tuple with high multiplicity. For example, if both the correct and erroneous l-tuples have multiplicity 3 (with default threshold $M = 2$), it is hard to decide whether we deal with a unique region (with coverage 6) or with two copies of an imperfect repeat (each with coverage 3). In the *N. meningitidis* project there were 1610 errors with multiplicity 3 and larger. Due to page limitation, the algorithm to correct high-multiplicity errors will be described elsewhere.

5. EULERIAN SUPERPATH PROBLEM

As we have discussed, the idea of the Eulerian path approach to SBH is to construct a graph whose edges correspond to l-tuples and to find a path visiting every edge of this graph exactly once.

Given a set of reads $S = \{s_1, \ldots, s_n\}$, define the de Bruijn graph $G(S_l)$ with vertex set S_{l-1} (the set of all $(l-1)$-tuples from S) as follows. An $(l-1)$-tuple $v \in S_{l-1}$ is joined by a directed edge with an $(l-1)$-tuple $w \in S_{l-1}$, if S_l contains an l-tuple for which the first $l-1$ nucleotides coincide with v and the last $l-1$ nucleotides coincide with w. Each l-tuple from S_l corresponds to an edge in G.

If S contains the only sequence s_1, then this sequence corresponds to a path visiting each *edge* of G exactly once, an *Eulerian* path [16]. Finding Eulerian paths is a well-known problem that can be efficiently solved. The reduction from SBH to the Eulerian path problem described above assumes unit multiplicities of edges (no repeating l-tuples) in the de Bruijn graph (see below for a discussion on multiple edges). We usually assume that S contains a direct complement of every read. In this case, $G(S_l)$ includes reverse complement for every l-tuple and the de Bruijn graph can be partitioned into 2 subgraphs, one corresponding to a "canonical" sequence, and another one to its reverse complement.

With real data, the errors hide the correct path among many erroneous edges. The overall number of vertices in the graph corresponding to the error-free data from the NM project is 4,039,248 (roughly twice the length of the genome), while the overall number of vertices in the graph corresponding to real sequencing reads is 9,474,411 (for 20-mers). After the error-correction procedure this number is reduced to 4,081,857.

A vertex v is called a *source* if $indegree(v) = 0$, a *sink* if $outdegree(v) = 0$ and a *branching* vertex if $indegree(v) \cdot outdegree(v) > 1$. For the *N. meningitidis* genome, the de Bruijn graph has 502,843 branching vertices for original reads (for l-tuple size 20). Error corrections simplifies this graph and leads to a graph with 382 sources and sinks and 12,175 branching vertices. The error-free reads lead to a graph with 11173 branching vertices.

Since the de Bruijn graph gets very complicated even in the error-free case, taking into account the information about what l-tuples belong to the same reads (that was lost after the construction of the de Bruijn graph) helps us to untangle this graph.

A path $v_1 \ldots v_n$ in the de Bruijn graph is called a *repeat* if $indegree(v_1) > 1$, $outdegree(v_n) > 1$, and $outdegree(v_i) = 1$ for $1 \le i \le n-1$ (Fig. 2). Edges entering the vertex v_1 are called *entrances* into a repeat while edges leaving the vertex v_n are called *exits* from a repeat. An Eulerian path visits a repeat a few times and every such visit defines a pairing between an entrance and an exit. Repeats may create problems in fragment assembly since there are a few entrances in a repeat and a few exits from a repeat but it is not clear which exit is visited after which entrance in the Eulerian path. However, most repeats can be resolved by read-paths (i.e., paths in the de Bruijn graph that correspond to sequencing reads) *covering* these repeats. A read-path covers a repeat if it contains an entrance into this repeat and an exit from this repeat. Every covering read-path reveals some information about the correct pairings between entrances and exits. However, some parts of the de Bruijn graph are impossible to untangle due to long perfect repeats that are not covered by any read-paths. A repeat is called *a tangle* if there is no read-path containing this repeat (Fig. 2). Tangles create problems in fragment assembly since pairings of entrances and exits in a tangle cannot be resolved via the analysis of read-paths. To address this issue we formulate the following generalization of the Eulerian Path Problem:

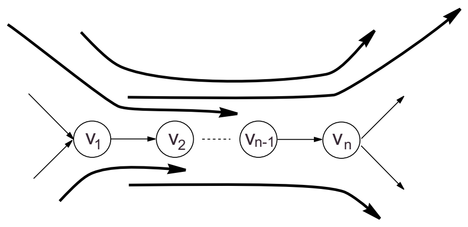

Figure 2: A repeat $v_1 \ldots v_n$ and a system of paths overlapping with this repeat. The uppermost path contains the repeat and defines the correct pairing between the corresponding entrance and exit. If this path were not present, the repeat $v_1 \ldots v_n$ would become a tangle.

Eulerian Superpath Problem. Given an Eulerian graph and a collection of paths in this graph, find an Eulerian path in this graph that contains all these paths as subpaths.

The classical Eulerian Path Problem is a particular case of the Eulerian Superpath Problem with every path being a single edge. To solve the Eulerian Superpath Problem we transform both the graph G and the system of paths $\mathcal{P}$ in this graph into a new graph G_1 with a new system of paths $\mathcal{P}_1$. Such transformation is called *equivalent* if there exists a one-to-one correspondence between Eulerian superpaths in $(G, \mathcal{P})$ and $(G_1, \mathcal{P}_1)$. Our goal is to make a series of equivalent transformations

$$(G, \mathcal{P}) \to (G_1, \mathcal{P}_1) \to \ldots \to (G_k, \mathcal{P}_k)$$

that lead to a system of paths $\mathcal{P}_k$ with every path being a single edge. Since all transformations on the way from $(G, \mathcal{P})$ to $(G_k, \mathcal{P}_k)$ are equivalent, every solutions of the Eulerian Path Problem in $(G_k, \mathcal{P}_k)$ provides a solution of the Eulerian Superpath Problem in $(G, \mathcal{P})$

Below we describe a simple equivalent transformation that solves the Eulerian Superpath Problem in the case when the graph G has no multiple edges. Let $x = (v_{in}, v_{mid})$ and $y = (v_{mid}, v_{out})$ be two consecutive edges in graph G and let $\mathcal{P}_{x,y}$ be a collection of all paths from $\mathcal{P}$ that include both these edges as a subpath. Define $\mathcal{P}_{\to x}$ as a collection of paths from $\mathcal{P}$ that end with x and $\mathcal{P}_{y \to}$ as a collection of paths from $\mathcal{P}$ that start with y. The *x,y-detachment* is a transformation that adds a new edge $z = (v_{in}, v_{out})$ and deletes the edges x and y from G (Fig. 3). This detachment

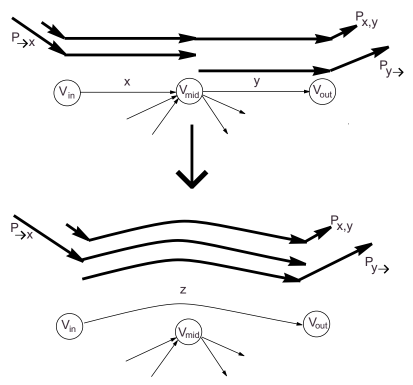

Figure 3: x, y-detachment is an equivalent transformation reducing the number of edges in the graph.

alters the system of paths $\mathcal{P}$ as follows: (i) substitute z instead of x, y in all paths from $\mathcal{P}_{x,y}$, (ii) substitute z instead of x in all paths from $\mathcal{P}_{\to x}$, and (iii) substitute z instead of y in all paths from $\mathcal{P}_{y\to}$. Informally, detachment bypasses the edges x and y via a new edge z and directs all paths in $\mathcal{P}_{\to x}$, $\mathcal{P}_{y\to}$, and $\mathcal{P}_{x,y}$ through z. Since every detachment reduces the number of edges in G, the detachments will eventually shorten all paths from $\mathcal{P}$ to single edges and will reduce the Eulerian Superpath Problem to the Eulerian Path Problem.

However, in the case of graphs with multiple edges, the detachment procedure described above may lead to non-equivalent transformations. In this case, the edge x may be visited many times in the Eulerian path and it may or may not be followed by the edge y on some of these visits. That's why, in case of multiple edges, "directing" all paths from the set $\mathcal{P}_{\to x}$ through a new edge z may not be an equivalent transformation. However, if the vertex v_{mid} has no other incoming edges but x, and no other outgoing edges but y, then x, y-detachment is an equivalent transformation even if x and y are multiple edges. In particular, detachments can be used to reduce every repeat to a single edge.

It is important to realize that even in the case when the graph G has no multiple edges, the detachments may create multiple edges in the graphs $G_1, \ldots, G_k$ (for example, if the edge (v_{in}, v_{out}) were present in the graph prior to the detachment procedure). However, such multiple edges do not pose problems, since in this case it is clear what *instance* of the multiple edge is used in every path (see below).

For illustration purposes, let's consider a simple case when the vertex v_{mid} has the only incoming edge $x = (v_{in}, v_{mid})$ with multiplicity 2 and two outgoing edges $y1 = (v_{mid}, v_{out1})$ and $y2 = (v_{mid}, v_{out2})$, each with multiplicity 1 (Fig. 4). In this case, the Eulerian path visits the edge x twice, in one case it is followed by $y1$ and in another case it is followed by $y2$. Consider an $x, y1$-detachment that adds a new edge $z = (v_{in}, v_{out1})$ after deleting the edge $y1$ and one of two copies of the edge x. This detachment (i) shortens all paths in $\mathcal{P}_{x,y1}$ by substitution of $x, y1$ by a single edge z and (ii) substitute z instead of $y1$ in every path from $\mathcal{P}_{y1\to}$. This detachment is an equivalent transformation if the set $\mathcal{P}_{\to x}$ is

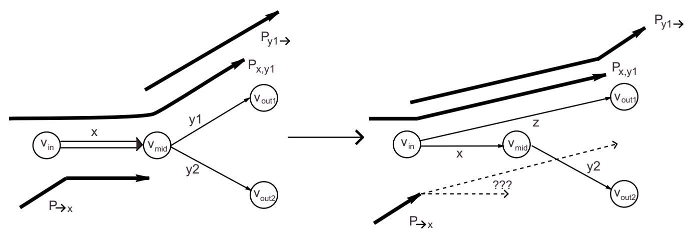

Figure 4: In the case when x is a multiple edge, $x, y1$-detachment is an equivalent transformation if $\mathcal{P}_{\to x}$ is empty. If $\mathcal{P}_{\to x}$ is not empty, it is not clear whether the last edge of a path $P \in \mathcal{P}_{\to x}$ should be assigned to z or to x.

empty. However, if $\mathcal{P}_{\to x}$ is not empty, it is not clear whether the last edge of a path $P \in \mathcal{P}_{\to x}$ should be assigned to the edge z or to the (remaining copy of) edge x.

To resolve this dilemma, one has to analyze every path $P \in \mathcal{P}_{\to x}$ and to decide whether it "relates" to $\mathcal{P}_{x,y1}$ (in this case it should be directed through z) or to $\mathcal{P}_{x,y2}$ (in this case it should be directed through x). By "relates" to $\mathcal{P}_{x,y1}$ ($\mathcal{P}_{x,y2}$) we mean that every Eulerian superpath visits $y1$ ($y2$) right after visiting P.

Two paths are called *consistent* if their union is a path again (there is no branching vertices in their union). A path P is *consistent* with a set of paths $\mathcal{P}$ if it is consistent with all paths in $\mathcal{P}$ and inconsistent otherwise (i.e. if it is inconsistent with at least one path in $\mathcal{P}$). There are three possibilities (Fig. 5)

- P is consistent with exactly one of the sets $\mathcal{P}_{x,y1}$ and $\mathcal{P}_{x,y2}$.
- P is inconsistent with both $\mathcal{P}_{x,y1}$ and $\mathcal{P}_{x,y2}$.
- P is consistent with both $\mathcal{P}_{x,y1}$ and $\mathcal{P}_{x,y2}$.

In the first case, the path P is called *resolvable* since it can be unambiguously related to either $\mathcal{P}_{x,y1}$ or $\mathcal{P}_{x,y2}$. If P is consistent with $\mathcal{P}_{x,y1}$ and inconsistent with $\mathcal{P}_{x,y2}$ then P should be assigned to the edge z after $x, y1$-detachment (substitute x by z in P). If P is inconsistent with $\mathcal{P}_{x,y1}$ and consistent with $\mathcal{P}_{x,y2}$ then P should be assigned to the edge x (no action taken). An edge x is called *resolvable* if all paths in $\mathcal{P}_{\to x}$ are resolvable. If the edge x is resolvable then the described x, y-detachment is an equivalent transformation after the correct assignments of last edges in every path from $\mathcal{P}_{\to x}$. In our analysis of the NM genome we found that 18026 among 18962 edges in the de Bruijn graph are resolvable. Although we defined the notion of resolvable path for a simple case in Fig. 3 when the edge x has multiplicity 2, it can be generalized for edges with arbitrary multiplicities.

The second condition (P is inconsistent with both $\mathcal{P}_{x,y1}$ and $\mathcal{P}_{x,y2}$) implies that the Eulerian Superpath Problem has no solution, i.e., sequencing data are inconsistent. Informally,

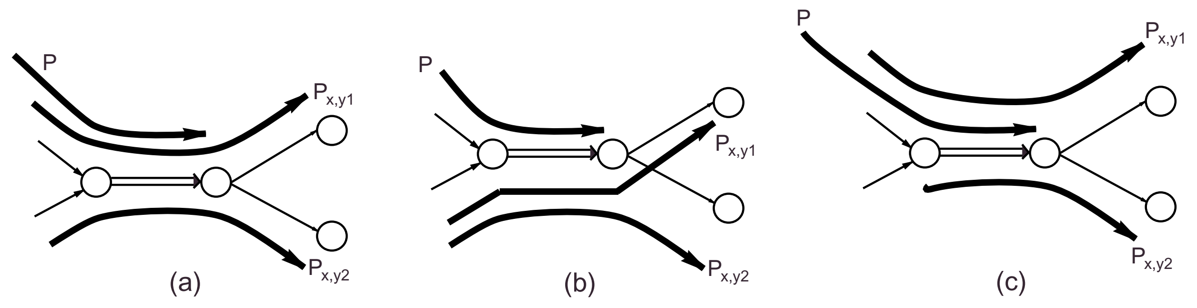

Figure 5: (a) P is consistent with $P_{x,y1}$, but inconsistent with $P_{x,y2}$; (b) P is inconsistent with both $P_{x,y1}$ and $P_{x,y2}$; (c) P is consistent with both $P_{x,y1}$ and $P_{x,y2}$.

in this case P, $\mathcal{P}_{x,y1}$ and $\mathcal{P}_{x,y2}$ impose three different scenario for just two visits of the edge x. After discarding the poor-quality and chimeric reads we did not encounter this condition in our analysis of the NM genome.

The last condition (P is consistent with both $\mathcal{P}_{x,y1}$ and $\mathcal{P}_{x,y2}$) corresponds to the most difficult situation and deserves a special discussion. If this condition holds for at least one path in $\mathcal{P}_{\to x}$, the edge x is called *unresolvable* and we postpone the analysis of this edge until all resolvable edges are analyzed. It may turn out that equivalent transformation of other resolvable edges will make the edge x resolvable. Fig. 6 illustrates that equivalent transformations may resolve previously unresolvable edges.

However, some edges cannot be resolved even after the detachments of all resolvable edges are completed. Such situations usually correspond to tangles and they have to be addressed by another equivalent transformations called a *cut*.

Consider a fragment of graph G with 5 edges and four paths $y3-x$, $y4-x$, $x-y1$ and $x-y2$, each path consisting of two edges (Fig. 7). In this case $\mathcal{P}_{\to x}$ consists of two paths $y3-x$ and $y4-x$ and each of those paths is consistent with both $\mathcal{P}_{x,y1}$ and $\mathcal{P}_{x,y2}$. In fact, in this symmetric situation, x is a tangle and there is no information available to relate any of path $y3-x$ and $y4-x$ to any of paths $x-y1$ and $x-y2$. Therefore, it may happen that no detachment is an equivalent transformation in this case. To address this problem, we introduce another equivalent transformation that affects the system of paths $\mathcal{P}$ and does not affect the graph G itself.

An edge $x=(v,w)$ is *removable* if (i) it is the only outgoing edge for v and the only incoming edge for w and (ii) x is either initial or terminal edge for every for every path $P \in \mathcal{P}$ containing x. An *x-cut* transforms $\mathcal{P}$ into a new system of paths by simply removing x from all paths in $\mathcal{P}_{\to x}$ and $\mathcal{P}_{x\to}$.

In the case of Fig.7, x-cut shortens the paths $x-y1$, $x-y2$, $y3-x$, and $y4-x$ to single-edge paths $y1$, $y2$, $y3$, and $y4$. It is easy to check that an x-cut of a removable edge is an equivalent transformation, i.e., every Eulerian superpath in $(G,\mathcal{P})$ corresponds to an Eulerian superpath in $(G,\mathcal{P}_1)$ and vice versa.

Cuts proved to be a powerful technique to analyze tangles that are not amenable to detachments. Detachments reduce such tangles to single unresolvable edges that turned out to be removable in our analysis of bacterial genomes. It allowed us to reduce the Eulerian Superpath Problem to the Eulerian Path Problem for all studied bacterial genomes.

Although detachments and cuts are sufficient to reduce the Eulerian Superpath Problem to the Eulerian Path Problem for the studied bacterial genomes, there is still a gap in the theoretical analysis of the Eulerian Superpath Problem in the case when the systems of paths is not amenable to neither detachments, nor

The idea of equivalent graph transformations for fragment assembly is conceptually similar to the idea of equivalent graph transformations for genome rearrangements [**?**]. We also emphasize that our equivalent transformation approach is very different from the graph reduction techniques for fragment assembly suggested in [8] and [10].

6. RESULTS

We tested EULER with real sequencing data from the *C. jejuni* (CJ) [13], *N. meningitidis* (NM) [12], and *L. lactis* (LL) [1] genomes (Table 1). These genomes were assembled either by Phrap (CJ and NM) or by gap4 (LL) software and required substantial finishing efforts to complete the assembly and to correct the assembly errors.

Orphan elimination and spectral alignment already provide a tenfold reduction in the error rate. However, further reductions in error rate are important since they simplify the de Bruijn graph and lead to the efficient solution of the Eulerian Superpath Problem. After the error correction is completed, the number of errors is reduced by a factor of $35-50$ making reads almost error-free.

To check the accuracy of the assembled contigs we *fit* [20] each assembled contig into the genomic sequence via local sequence alignment. A contig is assumed to be correct if it fits into a genomic sequence as a continuous fragment with a small number of errors. Inability to fit a contig into the genomic sequence with a small number of errors indicates that the contig is misassembled. For example, Phrap misassembles 17 contigs in the *N. meningitidis* sequencing project, each contig containing up to four fragments from different

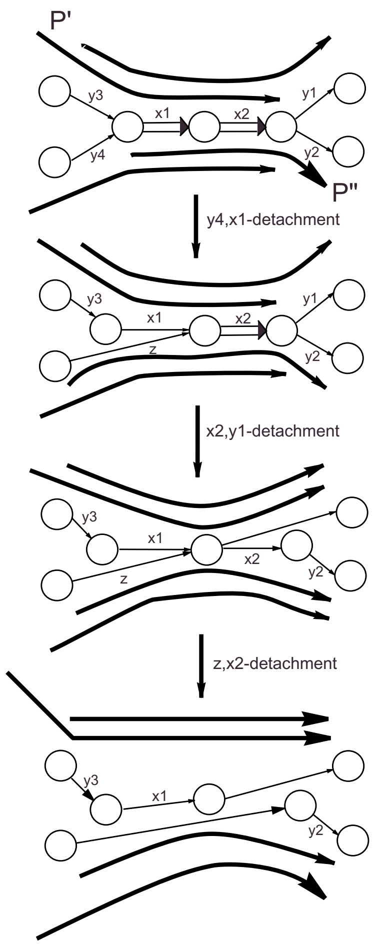

Figure 6: In the graph on the top, the edge $x2$ is not resolvable ($P' \in \mathcal{P}_{\to x2}$ is consistent with both $\mathcal{P}_{x2,y1}$ and $\mathcal{P}_{x2,y2}$) and the edge $x1$ is resolvable ($P'' \in \mathcal{P}_{x1\to}$ is consistent with $\mathcal{P}_{y4,x1}$ and inconsistent with $\mathcal{P}_{y3,x1}$). Therefore $y4, x1$-detachment is an equivalent transformation substituting $y4$ and $x1$ by z (second graph). This transformation makes $x2$ resolvable and opens a door for $x2, y1$-detachment (third graph). Finally, $z, x2$-detachment leads to a simplification of the original graph.

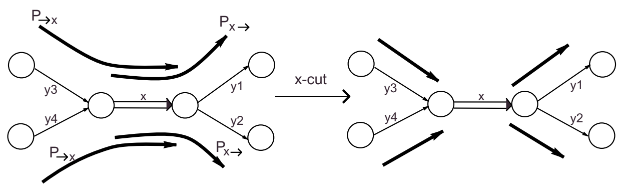

Figure 7: If x is a removable edge, then x-cut is an equivalent transformation shortening the paths in $\mathcal{P}$.

parts of the genome. We break misassembled contig into two or more fragments and fit each fragment separately into different locations in the genome.

To compare EULER with other fragment assembly algorithms, we ran Phrap, CAP3 and TIGR assemblers (default parameters) for CJ, NM, and LL sequencing projects (Table 2 and Figure 8). Every box in Figure 8 corresponds to a contig in NM assembly produced by these programs. Boxes in the IDEAL assembly correspond to islands in the read coverage. Boxes of the same shade show misassembled contigs, for example two identically shaded boxes in different places show the positions of contigs that were incorrectly assembled into a single contig. In some cases, a single shaded box shows a contig that was assembled incorrectly (i.e., there was a rearrangement inside this contig). The tangles are indicated by numbered boxes at the solid line showing the genome.

For example, in the CJ sequencing project, there are 2 tangles, one with multiplicity 3 and another with multiplicity 2. EULER produces 29 contigs and we can prove that it is an optimal assembly, i.e., no program can produce an assembly with a smaller number of contigs. The CJ sequencing project has 24 islands in the coverage and the overall multiplicity of tangles in this project is 5. Since all these tangles belong to different islands, the lower bound for the number of contigs in any valid assembly is 24+5=29. An important advantage of EULER is that it suggests five PCR experiments that resolve all tangles and generate the IDEAL assembly. This feature is particularly important for the LL project since $\approx$ 50 PCR experiments would reduce the number of reported contigs by a factor of tenfold.

The *C. jejuni* sequencing project is relatively simple due to an unusually low number of long perfect repeats. However, even in this relatively simple case, Phrap, CAP3, and TIGR made assembly errors. This genome contains only four long repeated sequences with similarity 90% and higher. However, only two of these four repeats contain long perfect sub-repeats (tangles) and cannot be resolved even in theory. EULER resolves two other repeats and breaks the contigs at the positions of tangles to avoid potential assembly errors. *N. meningitidis* contains hundreds of repeats, some of them very long. Among these repeats 31 are tangles (each repeating four times on average) and extensive finishing work was required to untangle them.

L. lactis has a fewer number of repeats than *N. meningitidis* but it poses a different challenge: low coverage and high error rate. Although EULER successfully assembled this genome, further reduction in coverage and increase in error rate would "break" EULER. Table 3 and Fig. 8 study the question: "How low can EULER go (in coverage)?" by deleting some reads and running EULER on this reduced set of reads. These results indicate that EULER produces a better assembly with coverage 7-8 than other programs with the full coverage 9.7. The analysis of errors made by EULER in these simulated low coverage projects indicates that they are "reporting" rather than algorithmic errors. The problem is that in the low-coverage regions (e.g., regions with coverage 1), it is theoretically impossible to filter our the chimeric reads (some chimeric reads may look like perfectly legitimate reads). If such low-coverage regions are not reported by EULER (they are subject to finishing efforts anyway), then it becomes error-free again.

7. CONCLUSIONS

Finishing is a bottleneck in large-scale DNA sequencing. Of course, finishing is an unavoidable step to extend the islands and to close the gaps in read coverage. However, the existing programs produce much more contigs then the number of islands thus making finishing more complicated than necessary. What is worse, these contigs are often assembled incorrectly thus leading to time-consuming contig verification step. EULER bypasses this problem since the Eulerian Superpath approach transforms imperfect repeats into different paths in the de Bruijn graph. As a result, EULER does not even notice repeats unless they are long perfect repeats, i.e., when the corresponding paths cannot be separated. Tangles are theoretically impossible to resolve and therefore some additional biochemical experiments are needed to correctly position them.

Difficulties in resolving repeats led to the introduction of the double-barreled DNA sequencing and the breakthrough genome sequencing efforts at Celera [11]. The Celera assembler is a two-stage procedure that includes masking repeats at the overlap-layout-consensus stage with further ordering of contigs via the double-barreled information. It did not escape our attention that EULER has excellent scaling potential and the work on integrating EULER with the double-barreled data is now in progress. In fact, the complexity of EULER is mainly defined by the number of tangles rather than the number of repeats/length of the genome. We believe that assembly of some simple eukaryotic genomes with a small number of tangles may be even less challenging for EULER than the assembly of the *N. meningitidis* genome. EULER does not require masking the repeats but instead provides a clear view of the structure of the perfect repeats (tangles) in the genome. These tangles may be resolved either by double-barreled information or by additional PCR experiments. These PCR experiments do not even require sequencing of PCR products but can be done through simple length measurements of PCR products. Since only a few PCR primers are needed to resolve each tangle, this “on-demand” finishing method potentially may reduce the cost of DNA sequencing as compared to the double-barreled approach.

Since reliable programs for pure shotgun assembly of complete genomes are still unavailable, the biologists are forced to do time-consuming mapping, verification, and finishing experiments to complete the assembly. As a result, most bacterial sequencing projects today start from mapping, a rather time-consuming step. Of course, mapping provides some insurance against assembly errors but it is not a 100%-proof insurance and it does not come for free. The only computational reason for using mapping information is to correct assembly errors and to resolve some repeats. EULER promises to make mapping unnecessary for sequencing applications, since it does not make errors, resolve all repeats but tangles, and suggests very few PCR experiments to resolve tangles. The amount of experimental efforts associated with these ”on demand” experiments is much smaller than with mapping efforts.

8. ACKNOWLEDGEMENTS

We are indebted to David Harper and Julian Parkhill for their advice and feedback, as well as for providing sequencing data from different bacterial sequencing projects. We are grateful to Herbert Fleschner, Michael Fonstein, Dmitry Frishman, Eugene Goltsman, Alexander Karzanov, Ross Overbeek, Julian Parkhill, Steven Salzberg, and Alexei Sorokine for useful discussions and providing sequencing data.

9. REFERENCES

[1] A. Bolotin, S. Mauger, K. Malarme, S. D. Ehrlich, and A. Sorokin. Low-redundancy sequencing of the entire *lactococcus lactis* IL1403 genome. *Antonie van Leeuwenhoek*, 76:27–76, 1999.

[2] J. K. Bonfield, K. F. Smith, and R. Staden. A new DNA sequence assembly program. *Nucleic Acids Research*, 23:4992–4999, 1995.

[3] R. Drmanac, I. Labat, I. Brukner, and R. Crkvenjakov. Sequencing of megabase plus DNA by hybridization: theory of the method. *Genomics*, 4:114–128, 1989.

[4] D. Gordon, C. Abajian, and P. Green. Consed: a graphical tool for sequence finishing. *Genome Research*, 8(3):195–202, 1998.

[5] P. Green. Documentation for phrap. http://bozeman.mbt.washington.edu/phrap.docs/phrap.html, 1994.

[6] S. Hannenhalli and P. Pevzner. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing*, pages 178–189, 1995.

[7] X. Huang and A. Madan. CAP3: A DNA sequence assembly program. *Genome Research*, 9:868–877, 1999.

[8] R. Idury and M. Waterman. A new algorithm for DNA sequence assembly. *Journal of Computational Biology*, 2:291–306, 1995.

[9] Y. Lysov, V. Florent'ev, A. Khorlin, K. Khrapko, V. Shik, and A. Mirzabekov. DNA sequencing by hybridization with oligonucleotides. *Doklady Academy Nauk USSR*, 303:1508–1511, 1988.

[10] E. M. Myers. Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2:275–290, 1995.

[11] E. W. Myers, G. G. Sutton, A. L. Delcher, I. M. Dew, D. P. Fasulo, M. J. Flanigan, S. A. Kravitz, C. M. Mobarry, K. H. Reinert, K. A. Remington, E. L. Anson, R. A. Bolanos, H. H. Chou, C. M. Jordan, A. L. Halpern, S. Lonardi, E. M. Beasley, R. C. Brandon, L. Chen, P. J. Dunn, Z. Lai, Y. Liang, D. R. Nusskern, M. Zhan, Q. Zhang, X. Zheng, G. M. Rubin, M. D. Adams, and J. C. Venter. A whole-genome assembly of Drosophila. *Science*, 287:2196–2204, 2000.

[12] J. Parkhill, M. Achtman, K. D. James, S. D. Bentley, C. Churcher, S. R. Klee, G. Morelli, D. Basham, D. Brown, T. Chillingworth, R. M. Davies, P. Davis, K. Devlin, T. Feltwell, N. Hamlin, S. Holroyd, K. Jagels, S. Leather, S. Moule, K. Mungall, M. A. Quail, M. A. Rajandream, K. M. Rutherford, M. Simmonds, J. Skelton, S. Whitehead, B. G. Spratt, and B. G. Barrell. Complete dna sequence of a serogroup a strain of *neisseria meningitidis* Z2491. *Nature*, 404:502–506, 2000.

[13] J. Parkhill, B. Wren, K. Mungall, J. M. Ketley, C. Churcher, D. Basham, T. Chillingworth, R. M. Davies, T. Feltwell, S. Holroyd, K. Jagels, A. Karlyshev, S. Moule, M. J. Pallen, C. W. Penn, Q. A. Quail, M. A. Rajandream, K. M. Rutherford, A. H. van Vliet, S. Whitehead, and B. G. Barrell. The genome sequence of the food-borne pathogen *campylobacter jejuni* reveals hypervariable sequences. *Nature*, 403:665–668, 2000.

[14] I. Pe'er and R. Shamir. Spectrum alignment: efficient resequencing by hybridization. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 260–268, San Diego, USA, 2000.

[15] P. Pevzner. l-tuple DNA sequencing: computer analysis. *Journal of Biomolecular Structure and Dynamics*, 7:63–73, 1989.

[16] P. A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. The MIT Pres, 2000.

[17] J. Roach, C. Boysen, K. Wang, and L. Hood. Pairwise end sequencing: a unified approach to genomic mapping and sequencing. *Genomics*, 26:345–353, 1995.

[18] G. Sutton, O. White, M. Adams, and A. Kerlavage. TIGR assembler: A new tool for assembling large shotgun sequencing projects. *Genome Science & Technology*, 1:9–19, 1995.

[19] H. Tettelin, D. Radune, S. Kasif, H. Khouri, and S. L. Salzberg. Optimized multiplex PCR: efficiently closing a whole-genome shotgun sequencing project. *Genomics*, 62:500–507, 1999.

[20] M. Waterman. *Introduction to Computational Biology*. Chapman Hall, 1995.

[21] J. Weber and G. Myers. Whole genome shotgun sequencing. *Genome Research*, 7:401–409, 1997.

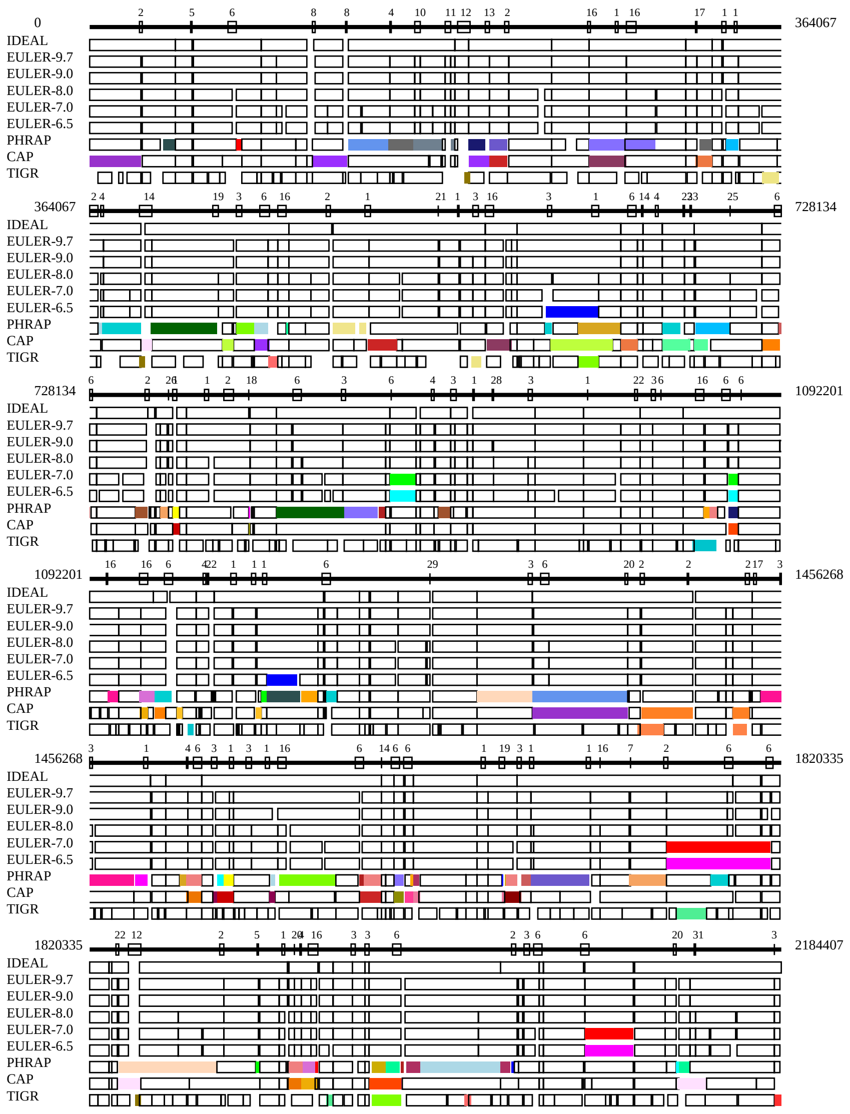

Figure 8: Comparative analysis of EULER, Phrap, CAP and TIGR assemblers as well as EULER on real and simulated low-coverage sequencing data (*N. meningitidis* sequencing project). Shaded boxes correspond to assembly errors.

Table 1: Assembly of reads from *C. jejuni* (CJ), *N. meningitidis* (NM), and *L. lactis* (LL) sequencing projects.

Project	CJ	NM	LL
genome length	1,641,481	2,184,406	2,365,590
average read-length	502	400	568
# of islands in reads coverage	24	79	6
coverage	10.3	9.7	6.4
# of reads	33708	53263	26532
# of poor-quality reads	431	251	128
# of chimeric reads	611	874	935
error rate	1.6%	1.2%	2.1%
# of errors per read (on average)	8.0	4.8	11.9
# of errors per read after			
orphan elimination/spectral alignment	0.84	0.36	1.4
correcting high-multiplicity errors	0.41	0.30	0.78
filtering poor-quality and chimeric reads	0.17	0.11	0.32
after error corrections			
coverage	10.1	9.0	6.3
# of reads	32666	52138	25469
# consistent errors per read	0.16	0.10	0.27
# inconsistent errors per read	0.01	0.01	0.05
% of corrected errors	97.9%	97.7%	97.3%
before solving the Eulerian Superpath Problem			
# of vertices in the de Bruijn graph ($l = 20$)	3,197,687	4,081,857	4,430,803
# of branching vertices ($l = 20$)	2132	12175	4873
after solving the Eulerian Superpath Problem			
# of vertices in the de Bruijn graph ($l = 20$)	126	999	148
# of branching vertices ($l = 20$)	30	617	124
# of sources/sinks ($l = 20$)	96	382	24
# of edges	74	1028	63
# of connected single-edge components ($l = 20$)	26	112	48
# of connected components ($l = 20$)	33	122	62
# of tangles	2	31	16
overall multiplicity of tangles	5	126	61
maximum multiplicity of a tangle	3	21	9
running time (hours)	3	5	6
8 CPU 9Gb Sun Enterprise E4500/E5500			

Table 2: Comparison of different software tools for fragment assembly. IDEAL is an imaginary assembler that outputs the collection of islands in clone coverage as contigs. In the IDEAL column the number in parenthesis shows the overall multiplicity of tangles.

		IDEAL	EULER	Phrap	CAP3	TIGR assembler
CJ	# of contigs (# of misassembled contigs)	24(5)	29(0)	33 (2)	54 (3)	> 300 (> 10)
	coverage by contigs	99.5%	96.7%	94.0%	92.4%	90.0%
	coverage by misassembled contigs	-	0.0%	16.1%	13.6%	1.2%
NM	# of contigs (# of misassembled contigs)	79(126)	149 (0)	160 (17)	163 (14)	> 300(9)
	coverage by the contigs	99.8%	99.1%	98.6%	97.2%	87.4%
	coverage by misassembled contigs	-	0.0%	10.5%	9.2%	1.3%
LL	# of contigs (# of misassembled contigs)	6(61)	58 (0)	62 (10)	85 (8)	245 (2)
	coverage by the contigs	99.9%	99.5%	97.6%	97.0%	90.4%
	coverage by misassembled contigs	-	0.0%	19.0%	11.4%	0.7%

Table 3: EULER's performance with reduced coverage (NM sequencing project).

# of reads	53263	49420	43928	38437	35690
coverage	9.7	9.0	8.0	7.0	6.5
# of contigs (# of misassembled contigs)	149(0)	152(0)	175(0)	182(2)	185(3)
coverage by the contigs	99.5%	99.1%	98.5%	95.5%	94.8%
coverage by the misassembled contigs	0.0%	0.0%	0.0%	4.1%	6.2%