BIOINFORMATICS ORIGINAL PAPER Vol. 23 no. 13 2007, pages 1573–1579
doi:10.1093/bioinformatics/btm153

Genome analysis

OSLay: optimal syntenic layout of unfinished assemblies

Daniel C. Richter[1,*], Stephan C. Schuster[2] and Daniel H. Huson[1]

[1]Center for Bioinformatics (ZBIT), Institute for Computer Science, Tübingen University, 72076 Tübingen, Germany and [2]Penn State University, Center for Comparative Genomics and Bioinformatics, University Park, PA 16802, USA

Received on January 16, 2007; revised on March 27, 2007; accepted on April 16, 2007

Advance Access publication April 26, 2007

Associate Editor: Martin Bishop

ABSTRACT

Summary: The whole genome shotgun approach to genome sequencing results in a collection of contigs that must be ordered and oriented to facilitate efficient gap closure. We present a new tool OSLay that uses synteny between matching sequences in a target assembly and a reference assembly to layout the contigs (or scaffolds) in the target assembly. The underlying algorithm is based on maximum weight matching. The tool provides an interactive visualization of the computed layout and the result can be imported into the assembly editing tool Consed to support the design of primer pairs for gap closure.

Motivation: To enhance efficiency in the gap closure phase of a genome project it is crucial to know which contigs are adjacent in the target genome. Related genome sequences can be used to layout contigs in an assembly.

Availability: OSLay is freely available from: http://www-ab.informatik.uni-tuebingen.de/software/oslay

Contact: drichter@informatik.uni-tuebingen.de

1 INTRODUCTION

In the prevalent whole genome shotgun (WGS) approach, a genome sequence is assembled from a collection of short sequences called *reads* (Sanger *et al*., 1982). The reads are obtained using automated sequencers based on the well-established Sanger method (Sanger *et al*., 1977) or using a sequencing-by-synthesis technique recently introduced by the company 454 (Margulies *et al*., 2005). Software is used to assemble reads together to obtain large, contiguous fragments called *contigs*. Reads are usually sequenced in pairs of known relative order and orientation. This *mate-pair* information is used to order and orient the contigs relative to each other, thus producing *scaffolds* or *supercontigs*.

Such a WGS project does not produce a *finished* (fully assembled) genome, but rather a collection of contigs or scaffolds and thus there remain gaps in the reconstructed sequence. The precise number of gaps depends on the level of sequencing coverage and also on features of the genome that determine how difficult the genome is to assemble, such as the number, size and fidelity of repeats or cloning bias, when Sanger sequencing is employed (Myers, 1999). The average read length is also an important parameter and the longer the reads, the easier the assembly problem becomes and the fewer gaps will be produced.

It is unclear whether new sequencing-by-synthesis techniques introduced by 454 and promised by Solexa Inc. and other companies will make the assembly problem easier. Although such methods produce substantially more sequence per dollar and are not affected by cloning bias, the read length obtainable is currently a lot shorter than what is obtainable by Sanger sequencing.

In *gap closure*, the goal is to produce a finished genome by using PCR to fill the gaps between the contigs of the assembly. For efficiency, this is usually done using pairs of primers located on different contig ends and then two simultaneous PCR reactions are performed that run 'toward each other'. This is a costly and time-consuming process and so the goal is to minimize the number of pairs that need to be considered. If the number of contigs is n and if no further information is given, then the number of pairs to be considered is $O(n^2)$. For any two contigs that are known to be adjacent in the target genome, it suffices to run one gap-closure PCR experiment for the two adjacent contig ends. Thus, ideally, if the order and orientation of all contigs produced by a WGS project were known, then the number of required PCR experiments would equal the number of gaps which is $O(n)$.

In WGS assembly, scaffolds describe the relative *layout* of sets of contigs obtained with the help of mate-pair information. The question arises whether additional information can be used to the layout the contigs or scaffolds. If the genome sequence of a related species is available, then one possibility is to order and orient contigs based on synteny of matched sequence between both genomes. We will refer to the genome sequence of a related species as a *reference* sequence, in the case of a finished sequence, or *reference assembly*, if the reference genome is unfinished.

A number of existing programs make use of synteny by comparing unfinished assemblies to existing protein sequences or peptide databases, connecting contig ends which are part of a single open reading frame [BACCardI (Bartels *et al*., 2005), PGAAS (Yu *et al*., 2002), CAAT-Box (Frangeul *et al*., 2004)]. Others use sequence markers on physical maps to confirm the order of contigs [MapLinker (Xu and Gordon, 2005)]. The web interface Projector2 (van Hijum *et al*., 2005) is a genome mapping tool for ordering prokaryotic assemblies determined by a template genome.

*To whom correspondence should be addressed.

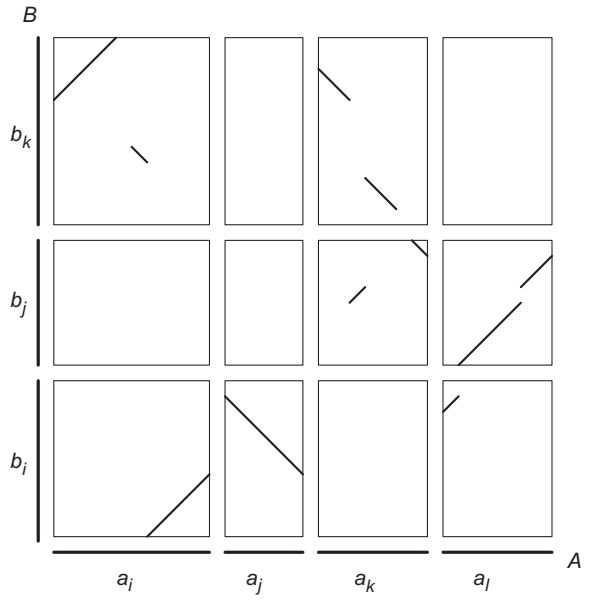

Fig. 1. Example of a comparison grid Z showing two assemblies A and B together with their a set of matches M. Cell z_{ii} contains a direct match whereas z_{ji} contains a reverse-complemented syntenic segment.

In this article, we present the *Optimal Syntenic Layout* (OSL) algorithm which aims at computing a *layout* (i.e. relative ordering and orientation) of a set of contigs or scaffolds based on syntenic information such as obtained by a sequence comparison of the contigs against a reference sequence or reference assembly. In addition to existing software tools, our approach enables to even use fragmented reference sequences (assemblies) such that the reference determines the order of the target contigs and vice versa.

We have implemented the algorithm in a computer program called OSLay (*Optimal Syntenic Layouter*). OSLay takes as input a *target assembly* (a set of contigs or scaffolds) to be laid out, and a reference sequence or assembly (also in the form of contigs or scaffolds) and computes an optimal layout of the target assembly, or if desired, of both the target assembly and the reference assembly. The original layout and the inferred layout are both displayed as enhanced, interactive dot plots. The layout can be output in a number of different file formats, including as an *ace* file directly importable into Consed (Gordon *et al.*, 1998) or as a list of predicted gap lengths between contigs.

OSLay has been successfully used to layout a number of assemblies and can also be to complement contigs generated by Sanger sequencing with sequence data obtained from sequencing-by-synthesis, as done in (Velicer *et al.*, 2006). OSLay is written in Java and installers for Linux/Unix, MacOS X and Windows XP are freely available from our website at: http://www-ab.informatik.uni-tuebingen.de/software/oslay

2 METHODS

In the following, we propose a formulation and algorithm for the OSL problem that aims at finding a syntenic layout of two assemblies that maximizes the number of pairs of extended local diagonals. A preliminary version of this approach was presented at the 2004 GCB conference (Friedrichs *et al.*, 2004). In the following, we will assume that both genomes used are presented as assemblies and will not always distinguish between target and reference, as the algorithm will treat them symmetrically. The case in which the reference genome is provided as a single sequence is a special case of this. The main idea of the OSL algorithm is to permute and flip contigs in the one assembly, while keeping the ordering and orientations in the other assembly fixed, so as to locally elongate the diagonals of sequence matches as much as possible.

Suppose we are given a target sequence G. An assembly $A=(a_1,\ldots,a_p)$ of G consists of a collection of contigs a_i that are putative substrings of G. (The algorithm can also be made to work if a_1 to a_p are scaffolds, rather than contigs, but we do not present the details here.)

Let G and H be two genomes with assemblies $A=(a_1,\ldots,a_p)$ and $B=(b_1,\ldots,b_q)$, respectively. A local sequence comparison of the two assemblies [e.g. with BLAST (Altschul *et al.*, 1990) or MUMmer (Kurtz *et al.*, 2004)] gives rise to a collection of matches $M=(m_1\,m_2,\ldots,m_r)$. A match m is specified as $(a, x_1, x_2, b, y_1, y_2, o)$, with $a\in A$, $1\le x_1<x_2\le|a|$, $b\in B$, $1\le y_1<y_2\le|b|$, and $o\in(-1,+1)$, where $|a|$ denotes the length of a and x_1, x_2, y_1, y_2 denote relative nucleotide positions within contig a and b. The interpretation of this is that m is a direct match between the interval with indices $[x_1,\ldots,x_2]$ in a and $[y_1,\ldots,y_2]$ in b, if $o=+1$ or a match in which the sequence of the second interval is reverse complemented, if $o=-1$.

Usually BLAST matches are rather short local matches that lie close to a common diagonal. To decrease complexity, any cluster of matches is replaced by a single *summarized* match m_s reflecting the total length and orientation of the cluster. We will use M_s to denote the set of summarized matches. We say that a match m_s is *informative*, if it is an 'overlap' or 'containment' match, but not an 'end-to-end' match. For our purposes, only informative matches are of interest, and this implies that the two assemblies should not be too *correlated*, that i.e. contig boundaries should not coincide (i.e. the contigs should not start and end at equivalent positions).

Our tool visualizes A, B and M together in a dot-plot or comparison grid Z (Fig. 1), where cell z_{ij} has width $|a_i|$ and height $|b_j|$. The set M_{ij} of all matches between a_i and b_j is displayed inside the cell z_{ij}. Match diagonals represent common syntenic segments of A and B. If a sequence segment exists in the reference assembly B that overlaps contig boundaries of A, i.e. if a subsequence of B matches parts of different contigs of A, then one can assume that these contigs should be located next to each other. In this case, an appropriate layout of the contigs may give rise to an extended diagonal in the comparison grid. By switching the roles of A and B, a contig layout for B can be found too.

2.1 The optimal syntenic layout problem

To be able to extend diagonals, one needs to know where summarized matches can be extended: If a summarized match m_s touches or comes close to the side of a contig of A, an 'anchor point' of the cell side is defined called a *connector* $c=(y,w,o)$. It has height y that represents the position where m_s touches (or would touch) the side of the cell, a weight w which is the length of m_s and finally an orientation o representing the orientation of m_s, i.e. whether m_s has a 45 or -45° slope.

Consider two cells, z_{ij} and z_{kj} in the same row. Let C_{ij}^{right} be the set of all right connectors associated with z_{ij} and C_{kj}^{left} be the set of all left connectors associated with z_{kj}. We say that two connectors $c=(y,w,o)\in C_{ij}^{\text{right}}$ and $c'=(y',w',o')\in C_{kj}^{\text{left}}$ form a *local diagonal extension*, if c' extends c, i.e. if $y\sim y'$ and $o=o'$. We define the weight of such an extension as $w+w'-|h-h'|$, that is, the sum of weights of the involved matches, penalized by their height difference (Fig. 2a).

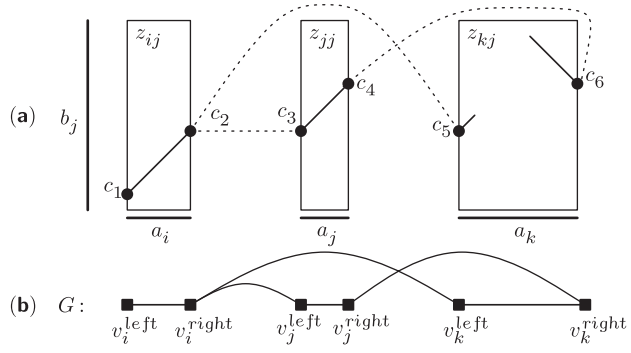

Fig. 2. (**a**) Three cells of the comparison grid Z with connectors c_i guiding the ordering process: Dotted lines represent possible side-by-side connections between the three contigs a_i, a_j, $a_k \in A$. (**b**) Layout graph G which is induced by local extensions: for each possible connector connection, a weighted edge is added to G between two contig side nodes v_i^{δ}, v_j^{ε} with $i \neq j$ and δ, $\varepsilon \in$ (left, right). Every pair of contig side nodes deriving from the same contig are incident to a contig edge.

Every possible diagonal extension between any two contig sides is assigned a weight. It is important to mention that connectors either from rows or from columns are considered but not both at the same time. Therefore, the two problems of determining a layout for the target assembly or a layout for the reference assembly are independent of each other.

The check for consistent diagonal extensions assumes that for all contigs (whole columns and rows, respectively) of the target assembly every possible combination of sides $\delta, \varepsilon \in$ (left, right) is examined. Given two columns (rows) a_{ia_i} and b_j, we define the score of matching the δ-side of a_{ia_i} to the ε-side of b_j as the sum of weights of all local diagonal extensions obtained for cells contained in the two columns (rows).

To introduce the OSL problem, a layout of assembly A is defined as a signed permutation

$$\pi : (1, \ldots, p) \mapsto (\pm\pi(1), \ldots, \pm\pi(p)),$$

where $|\pm\pi(i)|$ denotes the position of contig i in the ordering and $\text{sign}(\pm\pi(i))$ denotes its orientation, i.e. whether i was flipped, or not.

The *OSL* problem can now be formulated:

The OSL problem is to determine a layout π *of A that maximizes the sum of scores of local diagonal extensions.*

In terms of the comparison grid, this corresponds to finding an ordering and orientation of the columns (or rows) of the grid such that the sum of scores of pairs of adjacent column sides (or row sides, respectively) is maximized.

2.2 The OSL graph

A layout graph $G = (V,E,\omega)$ is defined with vertex set V, edge set E and an edge weight function ω which assigns a positive weight to every edge in the graph.

For each column a_i of the grid Z, we define two nodes v_i^{left} and v_i^{right} that correspond to the left and right sides of the column. Consider a pair of nodes v_i^{δ} and v_j^{ε} representing two different columns a_i^{δ} and a_j^{ε}, with δ, $\varepsilon \in$ (left, right). We define an edge e between the two nodes v_i^{δ} and v_j^{ε}, if the score S of matching the δ-side of column a_i with the ε-side of column a_j is >0. In this case, we set the weight $\omega(e)$ equal to S (Fig. 2).

Given a layout π of A, we say that an edge $e \in E$ between two nodes $v_i^{\delta} \in V$ and $v_j^{\varepsilon} \in V$ is *realized*, if the δ-side of v_i is adjacent to the ε-side of v_j in the layout. In other words, we require that $|\pi(i) - \pi(j)| = 1$ and the orientations are appropriate so that the two sides under consideration are next to each other.

By definition of the graph G, we have:

Lemma 1
The OSL problem is equivalent to finding a layout π *of A that maximizes the sum of weights of all realized edges in the graph G.*

This implies:

Lemma 2
The OSL problem is NP-hard.

Proof. We construct a reduction of the TSP problem with all distances in {1,2} (Garey and Johnson, 1979). Given a set $C = (c_1, \ldots, c_p)$ of cities and a distance $D(i, j) \in (1,2)$ for every pair of cities. Construct two assemblies, $A = (a_1, \ldots, a_p)$, where a_i represents city c_i, and $B = (b_1, \ldots, b_q)$, with $q = 2p^2$. For any two numbers $1 \leq i,j \leq p$ set $k = (i-1)p + j \in (1, \ldots, p^2)$ and consider two cells z_{ik} and z_{jk}. Place a positive line segment that touches the right side of z_{ik} and another that touches the left side of z_{jk}, so that they form an extension of weight 1. Also, place two such segments touching the left side of z_{ik} and right side of z_{jk}, respectively. If $D(i, j) = 2$, then, additionally, set $k' = p^2 + k$ and place four such line segments in cells $z_{ik'}$ and $z_{jk'}$, too. Hence, if a_i and a_j are adjacent in the obtained layout, then 1 or 2 will be contributed to the score, depending on whether the corresponding edge in the input graph has weight 1 or 2, respectively. Given this construction, the set of optimal layouts of A corresponds precisely to the set of all optimal tours of the cities.

2.3 The OSL algorithm

The problem of finding a *maximum weight matching* in $G = (V,E,\omega)$ can be solved efficiently (Gabow, 1976). Consider such a matching $U \subseteq E$. For the following discussion, we add a set F of additional *contig edges* to the graph: For every pair of nodes $v_i^{\text{left}}, v_i^{\text{right}} \in V$ coming from the same contig a_i, we add an edge connecting these two nodes. Consider the graph $G' = (V, U \cup F)$ containing only the matching edges and the contig edges. As the contig edges themselves form a matching, the graph G consists only of paths and even-length cycles.

If the graph contains no cycles, then a solution of the OSL problem is obtained simply by laying out the contigs of A in any way that preserves the layout induced by the chains. If the graph contains one or more cycles, then each such cycle must be broken by removing a matching edge of minimum weight. In this way, each cycle loses less than half of its total weight. Because there may exist another solution that does not involve cycles, in the worst case, breaking cycles in this way may produce a solution that has only half the weight of an optimal solution. Here is a summary of the algorithm:

Algorithm 1
Input: Assemblies A and B, and matches M
Output: A layout for A
Construct the graph $G = (V,E,\omega)$*, as described above*
Compute a maximum matching $U \subseteq E$
Let F be the set of all contig edges
Construct $G' = (V, U \cup F, \omega)$
For each cycle C inG:
    Delete the smallest weight edge in $C \cap U$
Greedily link all resulting paths into one path visiting all nodes
Traverse the chain and report the resulting layout.

We have shown the following result:

Theorem 1
Algorithm 1 computes a 2-approximation for the OSL problem.

Note that if the graph G does not contain cycles, the obtained result is optimal. In practice, such cycles will occur only rarely and thus the algorithm will often produce an optimal result.

3 IMPLEMENTATION

We have implemented the OSL algorithm in a program called *OSLay*. Our intention was to produce an interactive tool that can be integrated into a typical assembly pipeline. OSLay's main features are:

- an interactive user interface for exploring and visualizing the data,
- applicability to either prokaryotic and eukaryotic genomes,
- layout of a target assembly based on a given reference genome, or the layout of two assemblies simultaneously,
- integration into the assembly and finishing pipeline of the assembler Phrap and viewing software Consed by providing an output directly usable for easy primer picking,
- several possibilities to filter and adapt data such as trimming of unmatched contig ends, and
- detection of recombinations or putative misassembles and handling of typical contig-end artifacts.

3.1 Basic design

OSLay is written in Java and uses the sequence visualization engine provided by *CGViz* (Delgado-Friedrichs *et al.*, 2003). The program provides an enhanced dot-plot visualization of the comparison of two assemblies both before and after layout. The program runs well on small and medium size genomes (~200 Mb).

OSLay takes three files as input: two FASTA files containing the contigs of two assemblies as DNA sequences, referred to as the target and reference assemblies, and the corresponding matches file which is previously computed using BLAST or MUMmer. Repeat filtering [e.g. with RepeatMasker (Smit *et al.*, 1996–2004)] is required for large eukaryotic genomes.

3.2 Visualization

After parsing the data, three views are generated: the first view (*original data view*) displays all contigs sorted by their lengths and all matches. Horizontal and vertical thin blue lines indicate the contig borders that define the comparison grid.

The second view shows the same match distribution as the original data view with one restriction: only summarized matches which give rise to connectors are displayed. If matches touch (or almost touch) contig sides in the raw data view, a connector is placed at the concerned contig border. Connectors are colored green or red if they are placed on the vertical or horizontal contig borders, respectively.

Finally, the third view (*syntenic layout view*) depicts the result of running the OSL algorithm. In the resulting layout, contigs are ordered and oriented to form new *supercontigs*. Supercontigs are surrounded by framed boxes and display the connectors. Ideally, one or several extended match diagonals involving single contigs and covering most of the genome will be obtained. All visualizations can be interactively explored using OSLay's dot-plot navigation tools.

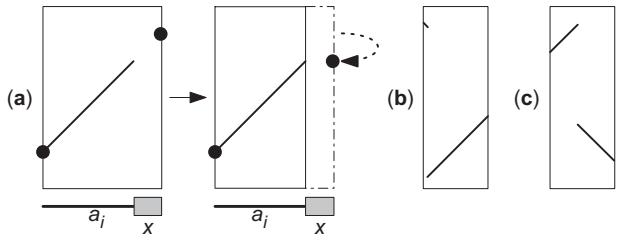

Fig. 3. Some additional features of OSLay. (**a**) Trimming of unmatched contig ends: contig a_i contains a piece of inserted foreign DNA x that does not match anything in the other assembly. Ignoring x, the connector is set to a different height. (**b**) If the short match at the top left side is due to misassembly, then it should be ignored. (**c**) A single cell containing a broken match may either indicate a recombination or a misassembly depending on the type of input of data.

A number of parameters can be set to govern the match summarizing process, the setting of connectors or the syntenic ordering. Displayed matches, cells, connectors, etc. can all be interactively queried to obtain information such as their id, length, type of match, etc.

3.3 Additional features and enhancements

In practice, difficulties arise due to assembly artifacts present at the ends of contigs (which presumably may also cause the assembly to break into contigs in the first place), and also due to evolutionary events that differentiate the target and reference genomes. Three of the most common problems are as follows:

(1) *Inserts cause unmatched regions in target contigs*. If an insertion of foreign DNA (e.g. phage DNA) took place in the target genome, but not in the reference genome, then no sequence matches will be found in the corresponding region in the dot plot. In particular, if the insert is located at a contig end in the target assembly, then the construction of a contig layout might be misled and some possible local extension between connectors might be missed. To address these complications, OSLay provides an option to ignore unmatched contig ends by setting connector positions directly to the locations where matches actually end and not where they are projected to end (Fig. 3a).

(2) *Bad sequence quality, artifacts or misassemblies*. Undesirable artifacts like obvious misassamblies at contig ends can prevent a successful contig layout because they give rise to ambiguous connector assignments. OSLay provides an option to ignore these misleading matches. In Figure 3b, the short match at the top corner can be ignored in further computations when setting connectors.

(3) *Repetitive regions*. Another observation is that repetitive regions in the reference genome that repeatedly map to a contig end of the target genome often complicate further computation. Because every summarized match

located near contig sides gives rise to a connector, as a consequence, too many misleading connectors are generated. To get rid of unusable connectors based on repeats, the user is able to filter repeats on both axes.

OSLay can selectively display recombinations and/or misassembles. This feature is useful when comparing two related assemblies or when assembling the same genome with two different assemblers. Recombinations are visible as 'broken matches' (Fig. 3c) in which at least two matches show different slopes in a single cell. When analyzing the result of an assembler–assembler comparison, this configuration indicates that one assembler has assembled a contig differently with respect to the other assembler. This type of situation is not modified by the OSL algorithm.

As already mentioned, instead of contigs OSLay is able to sort scaffolds as well: contig sequences contained in a scaffold can be concatenated. Gaps between contigs are filled with Ns if the gap size is known. Using this as input for BLAST and given a suitable reference genome, one will be able to sort the scaffolds. Since OSLay offers a parameter to adjust the allowed gap distance between matches situated near a putative diagonal, larger gaps between contigs contained in scaffolds do not complicate the sorting procedure.

3.4 Output

In addition to the visualized results, OSLay provides the user with several output files that can be used for various subsequent analyses:

(1) A *supercontig list* of all computed supercontigs and contained contigs.

(2) A *multi-fasta file* containing all supercontig DNA sequences. Each supercontig is a single record in a multi-fasta file. Single contig sequences are automatically ordered, reverse complemented (if required) and concatenated within each supercontig to reflect the computed contig layout. OSLay is able to estimate the gap distance between single contigs by computing the height difference between connectors linking two contig sides. These gaps are filled with 'N's. This provides an approximate estimation of the target genome size in relation to the reference genome. Additionally, the gap distance information (also provided as a single file) is valuable when it comes to primer design.

(3) A *contig mapping file* containing the position of every target contig in the reference genome.

(4) A *Phrap ace file* that is directly importable into the viewer and primer design software Consed. If the target assembly was produced using the Phrap assembler, an existing ace file can be taken as input and modified by OSLay so as to reflect the computed layout. All read coordinates and other related values are adapted, too. If the target assembly was not produced by Phrap, then OSLay can create an *ace* file from scratch. This simplifies the task of primer design, as contigs that are adjacent in the computed layout appear as neighbors in Consed. OSLay automatically assigns reverse-complemented contig (and read) sequence, when necessary.

4 RESULTS

Here, we show OSLay results for two different strains of the prokaryotic organism *Bdellovibrio bacteriovorus*. The HD100 strain is a predatory Gram-negative bacterium (Rendulic *et al*., 2004) that invades and consumes other Gram-negative bacteria. The HDHI strain was evolved from strain HD100 in a short time evolution experiment aimed at isolating a host independent mutant. Both genome sizes are ~3.78 Mb and differ only in a small amount of genes. (Rendulic *et al*., in preparation).

HD100 serves as the reference assembly and the set of 376 HDHI contigs is the target assembly to be ordered and oriented. To explore the performance of OSLay's layout algorithm, we considered several reference assemblies obtained at different sequencing stages of the HD100 genome. These consist of different number of reference contigs and thus give rise to different numbers of super contigs (Fig. 4a–d and Table 1). The HD100 assemblies are the product of a typical Sanger sequencing project whereas HDHI was sequenced with 12.97× coverage using Roche's sequencing-by-synthesis technology (unpublished data, Margulies *et al*., 2005). As both genomes are highly collinear, this experiment is an ideal example of a syntenic layout: The contigs of the target genome are almost completely laid out using the reference sequence.

Our results show that OSLay is able to order and orient the target contigs to obtain (partial) local extensions, i.e. elongation of local diagonals (Fig 4a–c). Even with a fragmented reference assembly consisting of 66 contigs (Fig. 4a), OSLay can significantly reduce the number of single contigs from 376 to 29 supercontigs, thus greatly simplifying the task of gap closure.

With increasing sequencing coverage, the number of computed supercontigs decreases, until only one super contig is left (Fig. 4d). Although nearly 90 contigs are not contained in the contig layout, the 286 sorted contigs cover ~99% of the summarized contig length. Only a set of contigs with a total length of ~34 kb (not shown in Table 1) remains unsorted.

5 DISCUSSION

The 'next-generation' sequencing technology is aimed at producing substantially more sequence in less time and for less money/megabase, but at the cost of decreased read lengths. Thus, genome sequences will continue to require WGS sequencing and assembly, however followed by a more demanding gap-closing phase, as the shorter read length results in a much higher number of contigs despite increased sequence coverage. As many more genome sequences of type strains become available, sequencing projects of closely related strains are increasingly performed and these profit from synteny-based contig layout, such as provided by OSLay.

The main application of OSLay is to produce a layout of contigs (or scaffolds) of a target assembly, given a reference

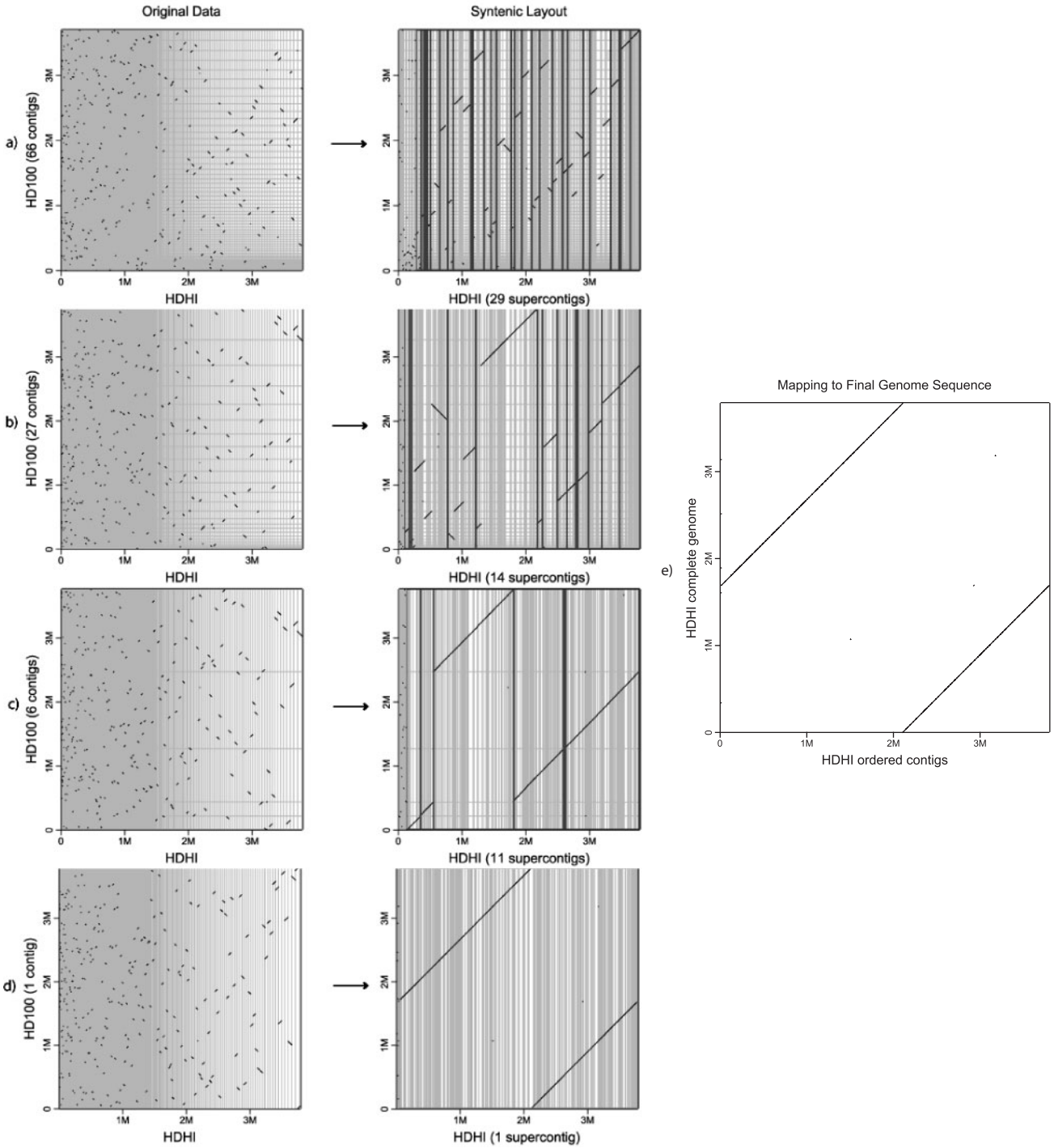

Fig. 4. Result views from four subsequent assembly phases of reference HD100 differing in the number of reference contigs (*y*-axis). Left column: original data view showing matches obtained by BLAST alignment. Right column: OSLay's computed syntenic layout with ordered and oriented contigs contained in supercontigs (black bars). The HDHI100 assembly (*x*-axis) consists of 376 contigs, which can be almost completely laid out (**d**) (see Table 1 for additional information). The last view shows a mapping of all ordered and concatenated contigs onto the finished genome sequence (e).

genome. While this is a trivial undertaking in case of a finished and closed reference genome, the OSLay approach becomes a powerful tool when used to layout a pair of assemblies simultaneously (e.g. when the reference genome itself is unfinished). Further, the program provides visual feedback on the scaffolding information and most importantly facilitates the import of syntenically ordered assemblies into Consed, a tool for assembly visualization and gap closure. This feature of

Table 1. OSLay statistic for four assembly stages (Fig. 4a–d)

Number of reference contigs	Number of supercontigs (contigs contained)	Total length of supercontigs (compared to total genome length)
66	29 (260)	3 513 114 bp (93%)
27	14 (274)	3 697 854 bp (98%)
6	11 (277)	3 704 402 bp (98%)
1	1 (286)	3 748 836 bp (99%)

Target assembly HD100 originally contains 376 contigs.

OSLay greatly simplifies the design of primer pairs for gap closure using PCR, as each amplicon spanning a gap now falls between the contig end sequences of two correctly ordered and oriented scaffolds.

A current trend in genome projects is to sequence one set of reads using Sanger sequencing and another set of reads using a sequencing-by-synthesis approach. The two approaches have different characteristics and so, when assembled separately, give rise to contigs with different contig boundaries, as both data require independent assembly programs. Each independent assembly can then be merged into a 'meta-assembly' using OSLay, with the side effect of visualizing possible misassemblies in either data set.

One drawback is that only closely related species can be used for ordering and sorting contigs. Our syntenic approach is not capable of sorting contigs if the used genomes or assemblies are derived from a more distant pair of species, which is not very surprising. The OSL algorithm works well for species from the same genus but usually has difficulties when using genomes from different orders or classes of the taxonomy.

The usage of mate-pairs (if available) is still the first choice to close a fragmented assembly. Thus, we plan to extend OSLay to take mate-pairs into account. This should help to increase the significance of contig links found by OSLay and to detect possible misassemblies.

OSLay has already been successfully applied to several recently sequenced microbial genomes at Penn State University, USA and at the Ludwig-Maximilian-University in collaboration with the Max-von-Pettenkofer Institute, Munich, Germany.

ACKNOWLEDGEMENTS

Funding for D.C.R. and D.H.H. was provided by the Deutsche Forschungsgemeinschaft (BIZ 1/1-2 & 1/1-3), D.C.R. and S.C.S. were supported in part by the Penn State University.

Conflict of Interest: none declared.

REFERENCES

Altschul,S.F. *et al*. (1990) Basic local alignment search tool. *J. Mol. Biol*., **215**, 403–410.

Bartels,D. *et al*. (2005) BACCardI – a tool for the validation of genomic assemblies, assisting genome finishing and intergenome comparison. *Bioinformatics*, **21**, 853–859.

Delgado-Friedrichs,O. *et al*. (2003) A meta-viewer for biomolecular data. *GI Jahrestagung*, **1**, 375–380.

Frangeul,L. *et al*. (2004) CAAT-Box, Contigs-Assembly and Annotation Tool-Box for genome sequencing projects. *Bioinformatics*, **20**, 790–797.

Friedrichs,O. et al. (2004) Syntenic Layout of Two Related Genomes. Proceedings of GCB 2004..

Gabow,H. (1976) An efficient implementation of Edmond's algorithm for maximum matching on graphs. *J. ACM*, **23**, 221–234.

Garey,M.R. and Johnson,D.S. (1979) *Computers and Intractability: a guide to the theory of NP-completeness*. W. H. Freeman and Company. New York, NY, USA.

Gordon,D. *et al*. (1998) Consed: a graphical tool for sequence finishing. *Genome Res*., **8**, 195–202.

Kurtz,S. *et al*. (2004) Versatile and open software for comparing large genomes. *Genome Biol*., **5**, R12.

Margulies,M. *et al*. (2005) Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, **437**, 367–380.

Myers,E.W. (1999) Whole-genome DNA sequencing. *IEEE Comput. Eng. Sci*., **3**, 33–43.

Rendulic,S. *et al*. (2004) A predator unmasked: life cycle of *Bdellovibrio bacteriovorus* from a genomic perspective. *Science*, **303**, 689–692.

Sanger,F. *et al*. (1977) DNA sequencing with chain-terminating inhibitors. *Biotechnology*, **24**, 104–108.

Sanger,F. *et al*. (1982) Nucleotide sequence of bacteriophage λ DNA. *J. Mol. Biol*., **162**, 729–773.

Smit,A.F.A. *et al*. (1996–2004) RepeatMasker Open-3.0. <http://www.repeatmasker.org>.

van Hijum,S. *et al*. (2005) Projector2: contig mapping for efficient gap-closure of prokaryotic genome sequence assemblies. *Nucleic Acids Res*., **33**, 560–566.

Velicer,G.J. *et al*. (2006) Comprehensive mutation identification in an evolved bacterial cooperator and its cheating ancestor. *Proc. Natl Acad. Sci. USA*, **103**, 8107–8112.

Xu,J. and Gordon,J.I. (2005) MapLinker: a software tool that aids physical map-linked whole genome shotgun assembly. *Bioinformatics*, **21**, 1265–1266.

Yu,Z. *et al*. (2005) PGAAS: a prokaryotic genome assembly assistant system. *Bioinformatics*, **18**, 661–665.