BIOINFORMATICS ORIGINAL PAPER

Vol. 27 no. 2 2011, pages 153–160
doi:10.1093/bioinformatics/btq646

Genome analysis

Advance Access publication November 18, 2010

Scoring-and-unfolding trimmed tree assembler: concepts, constructs and comparisons

Giuseppe Narzisi[1,*] and Bud Mishra[1,2]

[1]Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012 and [2]NYU School of Medicine, 550 First Avenue, New York, NY 10016, USA

Associate Editor: Joaquin Dopazo

ABSTRACT

Motivation: Mired by its connection to a well-known $\mathcal{NP}$-complete combinatorial optimization problem—namely, the Shortest Common Superstring Problem (SCSP)—historically, the whole-genome sequence assembly (WGSA) problem has been assumed to be amenable only to greedy and heuristic methods. By placing efficiency as their first priority, these methods opted to rely only on local searches, and are thus inherently approximate, ambiguous or error prone, especially, for genomes with complex structures. Furthermore, since choice of the best heuristics depended critically on the properties of (e.g. errors in) the input data and the available long range information, these approaches hindered designing an error free WGSA pipeline.

Results: We dispense with the idea of limiting the solutions to just the approximated ones, and instead favor an approach that could potentially lead to an exhaustive (exponential-time) search of all possible layouts. Its computational complexity thus must be tamed through a constrained search (Branch-and-Bound) and quick identification and pruning of implausible overlays. For his purpose, such a method necessarily relies on a set of score functions (*oracles*) that can combine different structural properties (e.g. transitivity, coverage, physical maps, etc.). We give a detailed description of this novel assembly framework, referred to as Scoring-and-Unfolding Trimmed Tree Assembler (SUTTA), and present experimental results on several bacterial genomes using next-generation sequencing technology data. We also report experimental evidence that the assembly quality strongly depends on the choice of the minimum overlap parameter *k*.

Availability and Implementation: SUTTA's binaries are freely available to non-profit institutions for research and educational purposes at http://www.bioinformatics.nyu.edu.

Contact: narzisi@nyu.edu

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Received on July 28, 2010; revised on October 19, 2010; accepted on November 13, 2010

1 INTRODUCTION

Since the pioneering work of Frederick Sanger in 1975, when he developed the basic DNA sequencing technology, Sanger chemistry has continued to be employed widely (Sanger *et al.*, 1977) in practically all large-scale genome projects. However, whole-genome sequence assembly (WGSA) pipelines in these projects have usually resorted to the *shotgun sequencing strategy* in order to reconstruct a genome sequence, despite the limitation that Sanger chemistry could only generate moderate-sized reads (around 1000 bp) with no location information. While many recent advances in sequencing technology has yielded higher throughput and lower cost, the limitations imposed by the read lengths (ranging between 35 and 500 bp) still plague the genomics science, forcing it to work with draft-quality, unfinished, genotypic and misassembled genomic data. The problem is, however, complicated by the presence of haplotypic ambiguities, base-calling errors and repetitive genomic sections. Recall that to obtain the input read data, the DNA polymer is first sheared into a large number of small fragments; and then either the entire fragment or just its ends are sequenced. The resulting sequences are then combined into a consensus sequence using a computer program: *DNA sequence assembler*. It is desired that the consensus has as small a base-level discrepancy with respect to the original DNA polymer as possible.

Researchers first approximated the shotgun sequence assembly problem as one of finding the shortest common superstring of a set of sequences (Tarhio and Ukkonen, 1988). Although this was an elegant theoretical abstraction, it was oblivious to what biology needs to make correct interpretation of genomic data. In fact, it misses the correct model for the assembly problem for at least three different reasons: (i) it does not model possible errors arising from sequencing the fragments; (ii) it does not model fragment orientation (the sequence source can be one of the two DNA strands); (iii) most importantly it fails in the presence of repeats in the genome. Faced with this theoretical computational intractability ($\mathcal{NP}$-complete), most of the practical approaches for genome sequence assembly were devised to use greedy and heuristic methods that, by definition, restrict themselves to find suboptimal solutions (see Kececioglu and Myers (1995)). Note that if the DNA was totally random then the overlap information would be sufficient to reassemble the target sequence and greedy algorithms would perform always well (Ma, 2009). However, this argument is mostly irrelevant, since the problem is complicated by the presence of various non-random structures, in particular in eukaryotic genomes (e.g. repeated regions, rearrangements, segmental duplications).

In the case of human genome, initially two unfinished draft sequences were produced by different methods, one by the International Human Genome Sequencing Consortium (IHGSC) and another by CELERA genomics (CG), with the published IHGSC assembly constructed by the program GigAssembler devised at

*To whom correspondence should be addressed.

the university of California at Santa Cruz (UCSC). As noted, in a recent article by Semple (2003): 'Of particular interest are the relative rates of misassembly (sequence assembled in the wrong order and/or orientation) and the relative coverage achieved by the three protocols. Unfortunately, the UCSC groups were alone in having published assessments of the rate of misassembly in the contigs they produced. Using artificial datasets, they found that, *on average 10% of assembled fragments were assigned the wrong orientation and 15% of fragments were placed in wrong order* by their protocol (Kent and Haussler, 2001). Two independent (more recent) assessments of UCSC assemblies have come to the similar conclusions'. Resolving these ambiguities requires the development of novel tools that can combine different technologies into one unified assembly framework, specifically combining short-range information (e.g. provided by sequence reads) together with single-molecule long-range information (e.g. provided by optical maps).

However, there is no unanimous agreement, at least within the computer science community, that this problem has exhausted all reasonable methods of attack. For instance, Karp (2003) observed, 'The shortest superstring problem [is] an elegant but flawed abstraction: [since it defines assembly problem as finding] a shortest string containing a set of given strings as substrings. The SCSP problem is $\mathcal{NP}$-hard, and theoretical results focus on constant-factor approximation algorithms … *Should this approach be studied within theoretical computer science*?' In contrast to the work in computational biology, there have now emerged examples within computer science, where impressive progress has been made to solve important $\mathcal{NP}$-hard problems *exactly*, despite their worst-case exponential time complexity: e.g. *Traveling Salesman Problem (TSP)*, *Satisfiability (SAT)*, *Quadratic Assignment Problem (QAP)*, etc. For instance, recent work of Applegate *et al.* (2001) demonstrated the feasibility of solving instances of TSP (as large as 85 900 cities) using *branch-and-cut*, whereas symbolic techniques in propositional satisfiability (e.g. DPLL SAT solver Davis *et al.*, 1962), employing systematic backtracking search procedure (in combination with efficient conflict analysis, clause learning, non-chronological backtracking, 'two-watched-literals' unit propagation, adaptive branching and random restarts), have exhibited the capability to handle more than a million variables.

Inspired by these lessons from theoretical computer science, a novel approach, embodied in SUTTA algorithm, was developed. In the process, several related issues were addressed: namely, developing better ways to dynamically evaluate and validate layouts, formulating the assembly problem more faithfully, devising superior and accurate algorithms, taming the complexity of the algorithms and finally, a theoretical framework for further studies along with practical tools for future sequencing technologies. Because of the generality and flexibility of the scheme (it only depends on the underlying sequencing technologies through the choice of score and penalty functions), SUTTA is capable, at least in principle, of agnostically adapting to various rapidly evolving technologies. It also allows concurrent assembly and validation of multiple layouts, thus providing a flexible framework that combines short- and long-range information from different technologies. The main aim of this article is to elaborate upon the mathematical, algorithmic and technical details of this method. The article also demonstrates SUTTA's feasibility through several *in silico* experiments using real paired and unpaired data from next-generation sequencing technology. Finally contig size and assembly quality are shown to be critically dependent on the minimum overlap parameter k.

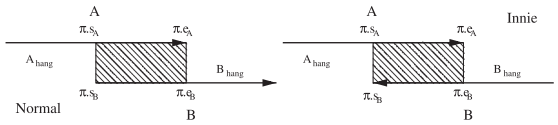

Fig. 1. Two possible overlaps: left overlap is *normal* (both reads pointing to the same forward direction) right overlap is *innie* (the second read B is reverse complemented and is pointing in the backward direction).

2 SYSTEM AND METHODS

2.1 The sequence assembly problem

In order to naturally derive SUTTA's method, it is best to start with a formulation of the assembly problem as a constrained optimization problem. For this purpose, SUTTA relies on the same notational framework (but a different approach of attack) as that first introduced by Myers (1995). For convenience, an essential set of definitions is summarized below.

The output of a sequencing project consists of a set of reads (or fragments) $F=\{r_1, r_2, \ldots, r_N\}$, where each read r_i is a string over the alphabet $\Sigma=\{A, C, G, T\}$. To each read is associated a pair of integers $(s_i, e_i), i \in [1, |F|]$ where s_i and e_i are, respectively, the starting and ending points of the read r_i in the reconstructed string R (to be generated by the assembler), such that $1 \leq s_i, e_i \leq |R|$. The order of s_i and e_i encodes the orientation of the read (whether r_i was sampled from Watson or Crick strand).

The overlaps (best local alignment) between each pair of reads may be computed using the Smith–Waterman algorithm (Smith and Waterman, 1981) with match, mismatch and gap penalty scores dependent on the errors introduced by the sequencing technology. Exact string matching is instead used for short read from next-generation sequencing, since they usually provide high coverage, thus allowing tolerance to an increased false negatives. Note also that by restricting to exact matches only, the time complexity of the overlap detection procedure is reduced from a quadratic to a linear function of the input size. [1] The complete description of an overlap π is given by specifying: (i) the substrings $\pi.A[\pi.s_A, \pi.e_A]$ and $\pi.B[\pi.s_B, \pi.e_B]$ of the two reads that are involved in the overlap; (ii) the offsets from the left-most and right-most positions of the reads $\pi.A_{hang}$ and $\pi.B_{hang}$; (iii) the relative directions of the two reads: normal (N), innie (I); (iv) a predicate $suffix_\pi(r)$ on a read r such that:

$$suffix_\pi(r) = \begin{cases} true \;\; iff \;\; \text{suffix of } r \text{ participates in the overlap } \pi \\ false \;\; iff \;\; \text{prefix of } r \text{ participates in the overlap } \pi \end{cases} \quad (1)$$

Figure 1 illustrates two possible overlaps (normal and innie). Note that a right arrow represents a read in forward orientation, conversely a left arrow represents a read that is reverse complemented.

DEFINITION 1 (Layout). *A layout L induced by a set of reads $F=\{r_1, r_2, \ldots, r_N\}$ is defined as:*

$$L = r_{j_1} \overset{\pi_1}{\rightleftharpoons} r_{j_2} \overset{\pi_2}{\rightleftharpoons} r_{j_3} \overset{\pi_3}{\rightleftharpoons} \cdots \overset{\pi_{N-1}}{\rightleftharpoons} r_{j_N}. \quad (2)$$

Informally a layout is simply an ordered sequence of reads connected by overlap relations. Note that the order of the reads in L is a permutation of the reads in F. The previous definition assumes that there are no *containments*[2]; without loss of generality, contained reads can be initially removed (in a preprocessing step) and then reintroduced later after the layout has been created. Among all the possible layouts (possibly, super-exponential in the

[1] See the Supplementary Material for more information about the overlapper.
[2] These are reads that are proper subsequences of another read.

number of reads), it is imperative to efficiently identify the ones that are consistent according to the following definition:

DEFINITION 2 (Consistency property). *A layout L is* **consistent** *if the following property holds for $i=2,\ldots,N-1$:*

$$\overset{\pi_{i-1}}{\rightleftharpoons} r_{j_i} \overset{\pi_i}{\rightleftharpoons} \text{ iff } suffix_{\pi_{i-1}}(r_{j_i}) \neq suffix_{\pi_i}(r_{j_i}). \quad (3)$$

The consistency property imposes a directionality to the sequence of reads in the layout. The estimated start positions for each read are given by:

$$sp_1 = 1,\ sp_i = sp_{i-1} + \pi_{i-1}.hang_{r_{j_{i-1}}} \ \text{ if } i > 1 \quad (4)$$

Consistent layouts must also satisfy the following property:

DEFINITION 3 (ϵ-valid layout). *Let $0 \leq \epsilon < 1$ be the maximum error rate of the sequencing process. A layout L is* **ϵ-valid** *if each read $r_i \in L$ can be aligned to the reconstructed string R with no more than $\epsilon|r_i|$ differences.*

Note that in practice the maximum error rate ϵ is used during the overlap computation to filter only detected overlaps between two reads r_1 and r_2 whose number of errors is no more than $\epsilon(|r_1|+|r_2|)$. Equipped with this set of definitions, the sequence assembly problem is formulated as follows:

DEFINITION 4 (Sequence Assembly Problem). *Given a collection of fragment reads $F=\{r_i\}_{i=1}^N$ and a tolerance level (error rate) ϵ, find a reconstruction R whose layout L is* **ϵ-valid**, **consistent** *and such that the following set of properties (oracles) are satisfied:*

- *Overlap-Constraint (O)): the cumulative overlap score of the layout is optimized.*
- *(Mate-Pair-Constraint (MP)): the distance between mate-pairs is consistent.*
- *(Optical-Map-Constraint (OM)): the observed distribution of restriction enzyme sites, C_{obs} is consistent with the distribution of experimental optical map data C_{src}.*

Each of these properties plays an important role in resolving problems that arise when real genomic data is used (e.g. data containing repeat-regions, rearrangements, segmental duplications, etc.). Note that, in the absence of additional information, among all possible layouts the minimum length layout is typically preferred (*shortest superstring*), although this choice is difficult to justify. As the genomic sequence deviates further and further from a random sequence, normally minimum length layout starts introducing various misassembly errors (e.g. compression, insertions, rearrangements, etc.). Note that, traditionally, assemblers have only optimized/approximated one of the properties [i.e. (O)], listed above, while checking for the others in a post-processing step. SUTTA, in contrast, views this problem as a constrained optimization problem with the feasible region, determined by the consistent layouts. It converts the group of constraints into appropriate score functions and uses them in combination to search for the optimal layout. This article illustrates these ideas with all but (OM) constraints, which will be described in a sequel. Finally note that this list of constraints is not exhaustive and it will likely change from year to year as new sequencing technologies become available and new types of long-range information become possible to produce. It is thus important to have an assembly framework that could dynamically and effortlessly adapt to the new technologies.

2.2 SUTTA algorithm

Traditional graph-based assembly algorithms use either the overlap-layout-consensus (OLC) or the sequencing-by-hybridization (SBH) paradigm, in which first the overlap/DeBruijn graph is built and the contigs are extracted later. SUTTA instead assembles each contig independently and dynamically one after another using the Branch-and-Bound (B&B) strategy. Originally developed for linear programming problems (Land and Doig, 1960), B&B algorithms are well-known searching techniques applied to intractable ($\mathcal{NP}$-hard), combinatorial optimization problems. The basic idea is to search the complete space of solutions. However, the caveat is that explicit enumeration is practically impossible (i.e. has exponential time complexity). The tactics honed by B&B is to limit itself to a smaller subspace that contains the optimum—this subspace is determined dynamically through the use of certain *well chosen* score functions. B&B has been successfully employed to solve a large collection of complex problems, whose most prominent members are TSP (traveling salesman problem), MAX-SAT (maximal satisfiability) and QAP (quadratic assignment problem).

The high level SUTTA pseudocode is shown in Algorithm 1. Here, two important data structures are maintained: a forest of double-trees (D-tree) $\mathcal{B}$ and a set of contigs $\mathcal{C}$. At each step, a new D-tree is initiated from one of the remaining reads in $\mathcal{F}$. Once the construction of the D-tree is completed, the associated contig is created and stored in the set of contigs $\mathcal{C}$. Next the layout for this contig is computed and all its reads are removed from the set of all available reads $\mathcal{F}$. This process continues as long as there are reads left in the set $\mathcal{F}$. Note that for the sake of a clear exposition, both the forest of D-trees $\mathcal{B}$ and the set of contigs $\mathcal{C}$ are kept and updated in the pseudocode; however, after the layout is computed, there is no particular reason to keep the full D-tree in memory, especially, where memory requirements are of concern.

Algorithm 1: SUTTA - pseudo code

```
Input: Set of N reads
Output: Set of contigs
B := ∅;                                   /* Forest of D-trees */
C := ∅;                                   /* Set of contigs */
F := ⋃_i^N {r_i};                         /* All the available reads */
while (F ≠ ∅) do
    r := F.getNextRead();
    if ( ¬isUsed(r) ∧ ¬isContained(r) ) then
        DT := create_double_tree(r);
        B := B ∪ {DT};
        Contig CTG := create_contig(DT);
        C := C ∪ {CTG};
        CTG.layout();                     /* Compute contig layout */
        F := F \ {CTG.reads};             /* Remove used reads */
    end
end
return C;
```

Finally, note that the proposed Algorithm 1 is input order dependent. SUTTA adopts the policy to always select the next unassembled read with highest occurrence as seed for the D-tree [also used by Taipan; Schmidt *et al.* (2009)]. This strategy has the property to minimize the extension of reads containing sequencing errors. However, empirical observations indicate that changing the order of the reads rarely affects structure of the solutions, as the relatively longer contigs are not affected. An explanation for this can be obtained through a probabilistic analysis of the data and a 0–1 law resulting from such an analysis.

2.3 Overlap score (weighted transitivity)

Like any B&B approach, a major component of SUTTA algorithm is the score function used to evaluate the quality of the candidate solutions that are dynamically constructed using the B&B strategy. SUTTA employs an overlap score based on the following argument. Large *de novo* sequencing projects typically have coverage higher than three, and this implies that frequently two overlapping regions of three consecutive reads in a region of correct layout share intersections. Events of this type are 'witness' to a transitivity relation between the three reads and they play an important role in identifying *true* positive[3] overlaps with high probability. Figure 2 shows an example of transitivity relation between three reads A, B and C. During contig layout construction, the overlap score uses the following basic principle to dynamically compute the score value of a candidate solution: if

[3]The two reads correctly originate from the same place in the genome.

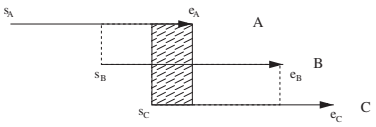

Fig. 2. Example of transitivity relation: the overlap regions between reads *AB* and *BC* share an intersection.

read *A* overlaps read *B*, and read *B* overlaps read *C*, SUTTA will score those overlaps strongly if in addition *A* and *C* also overlap:

$$\texttt{if}(\pi(A,B)\wedge\pi(B,C))\texttt{then}\{S(\pi(A,B,C))= S(\pi(A,B))+S(\pi(B,C))+(\pi(A,C)?S(\pi(A,C)):0)\} \quad (5)$$

Note that the score of a single overlap corresponds to the score computed by the Smith–Waterman alignment algorithm (for long reads) or exact matching (for short reads). Clearly the total score of a candidate solution is given by the sum of the scores along the overlaps that join the set of reads in the layout *L* plus the score of the transitivity edges (if any):

$$g(L)= \sum_{\substack{r_j\in L \\ j\in 2,\ldots,N-1}} S(\pi(r_{j-1},r_j,r_{j+1})) = \sum_{\pi_i\in O} S(\pi_i(r_{j_1},r_{j_2}))+\sum_{\pi_k\in T} S(\pi_k(r_{j_1},r_{j_2})), \quad (6)$$

where O and T are respectively the set of overlaps and transitivity edges (with respect to the set of reads, defined by the layout L) and $S(\pi)$ is the score (Smith–Waterman, exact match, etc.) for the overlap.

This step in SUTTA resembles superficially to the Unitig construction step in overlap-layout-consensus assembler, carried out by removing 'transitive' edges. However, unlike SUTTA, in the overlap-layout-consensus approach the weights of the overlaps are ignored in meaningfully scoring the paths. Since Unitig construction can be computationally expensive, large-scale assemblers like CELERA have adopted the *best-buddy* algorithm, where Unitigs are computed as chains of mutually unique best buddies (adjacent reads with best overlap between each other). Finally, it must be noted that the overlap scores are insufficient to resolve long repeats or haplotypic variations. The score functions must be augmented with constraints (formulated as reward/penalty terms) arising from mate-pair distance information or optical map alignments.

2.4 Node expansion

The core component of SUTTA is the B&B procedure used to build and expand the D-tree (create_double_tree() procedure in Algorithm 1). The high-level description of this procedure is as follows:

(1) Start with a random read (it will be the root of a tree; use only the read that has not been *used* in a contig yet, or that is not *contained*).

(2) Create RIGHT Tree: start with an unexplored leaf node (a read) with the best score value; choose all its non-contained *right*-overlapping reads (*Extensions*() procedure in Algorithm 2); filter out the set of overlapping reads by pruning unpromising directions (*Transitivity*(), *DeadEnds*(), *Bubbles*() and *MatePairs*() procedures in Algorithm 2); expand the remaining nodes by making them its children; compute their scores. (Add the 'contained' nodes along the way, while including them in the computed scores; check that no read occurs repeatedly along any path of the tree.) STOP when the tree cannot be expanded any further.

(3) Create LEFT Tree: symmetric to previous step.

Algorithm 2 presents the pseudocode of the expansion routine (details for each subroutine are available in the Supplementary Material). In this framework each path constructed using Algorithm 2 correspond to a possible layout of the reads for the current contig. Unlike the graph-based approaches (OLC and SBH), multiple paths/layouts are concurrently expanded and validated. Based on the branching strategy, two versions of SUTTA are

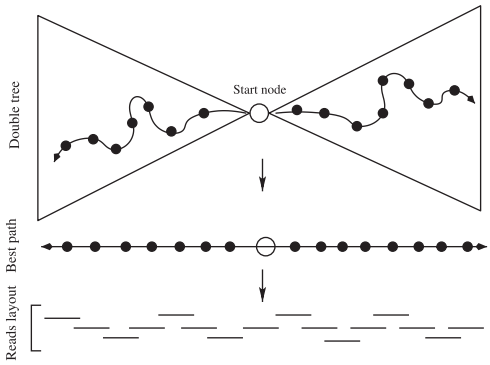

Fig. 3. Contig construction: (i) the D-tree is constructed by generating LEFT and RIGHT trees for the root node; (ii) best left and right paths are selected and joined together; (iii) the reads layout is computed for the set of reads in the full path.

available: if at the end of the pruning process there are still multiple directions to follow the branching is either terminated (*conservative*) or not (*aggressive*). In the aggressive case, the algorithm chooses the direction to follow with the highest local overlap. Algorithm 2 is applied twice to generate LEFT and RIGHT trees from the start read. Next, to create a globally optimal contig, the best LEFT path, the root and the best RIGHT path are concatenated together. Figure 3 illustrates the steps involved in the construction of a contig.

The amount of exploration and resource consumption is controlled by the two parameters K and T: K is the max number of candidate solution allowed in the queue at each time step, while T is the percentage of top ranking solutions compared with the current optimum score. At each iteration, the queue is pruned such that its size is always $\leq max(K, T|Q|)$, where $|Q|$ is the current size of the queue. Note that while K remains fixed at each iteration of Algorithm 2, the percentage of top ranking solutions dynamically changes over time. As a consequence, more exploration is performed when many solutions evaluate to nearly identical scores.

Algorithm 2: Node expansion

Input: Start read r_0, max queue size K, percentage T of top ranking solutions, dead-end depth W_{de}, bubble depth W_{bb}, mate-pair depth W_{mp}
Output: Best scoring leaf

$\mathcal{V}:=\emptyset$; /* Set of leaves */
$\mathcal{L}:=\{(r_0,g(r_0))\}$; /* Live nodes (priority queue) */
while $(\mathcal{L}\neq\emptyset)$ **do**
  $\mathcal{L}:=Prune(\mathcal{L},K,T)$; /* Prune the queue */
  $r_i:=\mathcal{L}$.popNext(); /* Get the best scoring node */
  $\mathcal{E}:=Extensions(r_i)$; /* Possible extensions */
  $\mathcal{E}^{(1)}:=Transitivity(\mathcal{E},r_i)$; /* Transitivity pruning */
  $\mathcal{E}^{(2)}:=DeadEnds(\mathcal{E}^{(1)},r_0,W_{de})$; /* Dead-end pruning */
  $\mathcal{E}^{(3)}:=Bubbles(\mathcal{E}^{(2)},r_0,W_{bb})$; /* Bubble pruning */
  $\mathcal{E}^{(4)}:=MatePairs(\mathcal{E}^{(3)},r_0,W_{mp})$; /* Mate pruning */
  if $(|\mathcal{E}^{(4)}|==0)$ **then**
    $\mathcal{V}:=\mathcal{V}\cup\{r_i\}$; /* r_i is a leaf */
  else
    for *(j=1* **to** $|\mathcal{E}^{(4)}|$*)* **do**
      $\mathcal{L}:=\mathcal{L}\cup\{(r_j,g(r_j))\}$;
    end
  end
end
return $\max_{r_i\in\mathcal{V}}\{g(r_i)\}$;

2.5 Search strategy

A critical component of any B&B approach is the choice of the search strategy used to explore the next subproblem in the tree. There are several

variations among strategies (with no single one being universally accepted as ideal), since these strategies' computational performance varies with the problem type. The typical trade-off is between keeping the number of explored nodes in the search tree low and staying within the memory capacity. The two most common strategies are Best First Search (BeFS) and Depth First Search (DFS). BeFS always selects among the live (i.e. yet to be explored) subproblems, the one with the best score. It has the advantage to be theoretically superior since whenever a node is chosen for expansion, a best-score path to that node has been found. However it suffers from memory usage problems, since it behaves essentially like a Breadth First Search (BFS). Also checking repeated nodes in a branch of the tree is computationally expensive (linear time). DFS instead always selects among the live subproblems the one with largest level (deepest) in the tree. It does not have the same theoretical guarantees of BeFS but the memory requirements are now bounded by the product of the maximum depth of the tree and the branching factor. The other advantage is that checking if a read occurs repeatedly along a path can be done in constant time by using the DFS interval schemes. For SUTTA, we use a combined strategy: using DFS as overall search strategy, but switching to BeFS, when choice needs to be made between nodes at the same level. This strategy can be easily implemented by ordering the set of live nodes $\mathcal{L}$ of Algorithm 2 using the following *precedence relation* between two nodes x and y:

$$x \prec y \quad \text{iff} \begin{cases} \text{depth}(x) > \text{depth}(y) \\ or \\ \text{depth}(x) == \text{depth}(y) \wedge \text{score}(x) > \text{score}(y) \end{cases}, \tag{7}$$

where depth is the depth of the node in the tree and score is the current score of the node (defined in section 2.3). Because BeFS is applied locally at each level, the score is optimized concurrently.

2.6 Pruning the Tree

Transitivity pruning: the potentially exponential size of the D-tree is controlled by exploiting certain specific structures of the assembly problem that permit a quick pruning of many redundant and uninformative branches of the tree—surprisingly, substantial pruning can be done only using local structures of the overlap relations among the reads. The core observation is that it is not prudent to spend time on expanding nodes that can create a suffix-path of a previously created path, as no information is lost by delaying the expansion of the *last* node/read involved in such a 'transitivity' relation. This scenario can happen every time there is a transitivity edge between three consecutive reads (see Fig. 2), and it is further illustrated in Figure 4 with an example. Suppose that $\langle A, B_1, B_2, \ldots, B_n \rangle$ are $n+1$ reads with a layout shown in Figure 4. The local structure of the D-tree will have node A with n children $B_1, B_2, \ldots, B_n$. However , since B_1 also overlaps $B_2, B_3, \ldots, B_n$, these nodes will appear as children of B_1 at the next level in the tree. So the expansion of nodes $B_2, B_3, \ldots, B_n$ can be delayed because their overlap with read A is enforced by read B_1. Similar argument holds for nodes $B_2, B_3, \ldots, B_n$. In the best scenario, this kind of pruning can reduce a full tree structure into a linear chain of nodes. Additional optimization can be performed by evaluating the children according to the following order ($h_1 \leq h_2 \leq \cdots \leq h_n$), where h_i is the size of the hang[4] for read B_i. This ordering gives higher priority to reads with higher overlap score. This explains how the *Transitivity*() procedure from Algorithm 2 is performed.

Zig-zag overlaps mapping: although based on a simple principle, the time complexity of the transitivity pruning is a function of how quickly it is possible to check the existence of an overlap between two reads (corresponding to the dashed arrows of Fig. 4). The general problem is the following: given the set of overlaps O (computed in a preprocessing step) for a set of reads F, check the existence of an overlap (or set of overlaps) for a pair of reads (r_1, r_2). The naive strategy that checks all the possible pairs takes time $O(n^2)$ where $n = |O|$. If a graph-theoretic approach is used,

[4]Size of the read portion that is not involved in the overlap.

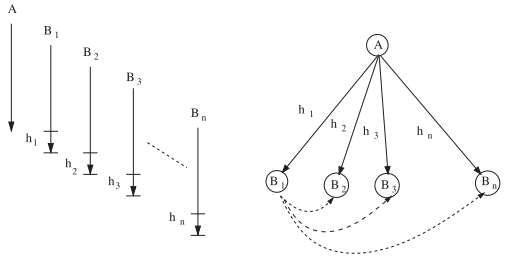

Fig. 4. Example of transitivity pruning: the expansion of nodes $B_2, B_3, \ldots, B_n$ can be delayed because their overlap with read A is enforced by read B_1.

by building the overlap-graph information (adjacency list), this operation takes time $O(l)$ where l is the size of the longest adjacency list in the graph. However, a much better [with $O(1)$ expected time] approach uses *hashing*. The idea is to build a hash table, in which a pair of reads is uniquely encoded to a single location of the table by using the following hash function:

$$H(a,b) = \frac{(a+b)(a+b-1)}{2} + (1-b), \tag{8}$$

where a and b are the unique identification numbers of the two reads. This is the well-known *zig-zag* function, which is the bijection often used in countability proofs. The number of possible overlaps $|H(a,b)|$ between two reads is always bounded by some constant c, which is a function of the read length, genome structure (e.g. number of simple repeats) and the strategy adopted for the overlap computation (Smith–Waterman, exact match, etc.). In practice, the costant c is never too large because, even when multiple overlaps between two reads are available (typically 4), only a small subset with a reasonably good score (i.e. above a threshold) is examined by the algorithm.

2.7 Lookahead

Mate-pairs: had the overlapping phase produced only true positive overlaps, every overlapping pair of reads would have been correctly inferred to have originated in the same genomic neighborhood, thus turning the assembly process to an almost trivial task. However, this is not the case—the overlap detection is not error free and produces false positive or ambiguous overlaps abundantly, specially when repeat regions are encountered. A potential repeat boundary between reads A, B and C is shown in Figure 5. Read A overlaps both reads B and C, but B and C do not overlap each other. Thus, the missing overlap between B and C is the sign of a possible repeat-boundary location, making the pruning decisions impossible. However, SUTTA's framework makes it possible to resolve this scenario by *looking ahead* into the possible layouts generated by the two reads, and keeping the node that generates the layout with the least number of unsatisfied constraints (i.e. consistent with mate-pair distances or restriction fragment lengths from optical maps).

SUTTA's implementation generates two subtrees: one for node B and the other for C (see Fig. 5). The size of each subtree is controlled by the parameter W_{mp}, the maximum height allowed for each node in the tree. The choice of W_{mp} is both a function of the size of the mate-pair library, local genome coverage and the genome structure. For genomes with short repeats, a small value for W_{mp} is sufficient to resolve most of the repeat boundaries, and can be estimated from a $k$mer analysis of the reads. However, some genomes have much higher complexity (family of LINEs, SINEs and segmental duplications with varying homologies), in that case a higher value of W_{mp} is necessary, but can be estimated adaptively. Once the two (or occasionally more) subtrees are constructed, the best path is selected based on the overlap score and the quality of each path is evaluated by a reward/penalty function corresponding to mate-pair constraints. For each node in the path, its pairing mate (if any) is searched to collect only those mate-pairs, crossing the connection point between the subtree and the full tree, which are then

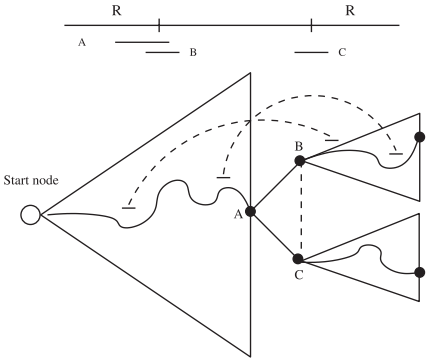

Fig. 5. Lookahead: the repeat boundary between reads B and C is resolved looking ahead in the subtree of B and C, and checking how many and how well the mate-pair constrains are satisfied.

scored by the following rule:

$$S_{MP}(r_1,r_2)=\begin{cases}1, & \text{iff}(l\in[\mu-\alpha\sigma,\mu+\alpha\sigma])\wedge(r_1\leftrightarrow r_2);\\ -1, & \text{iff}(l\notin[\mu-\alpha\sigma,\mu+\alpha\sigma])\wedge(r_1\leftrightarrow r_2);\\ -1, & \text{iff}\neg(r_1\leftrightarrow r_2);\\ 0, & \text{otherwise.}\end{cases} \quad (9)$$

Here l is the distance between the two reads in the layout, μ and σ are the mean and SD of the mate-pair library, α is a parameter that controls the relaxation of the mate-pair constrains (in the results, fixed at $\alpha=6$), and $r_1\leftrightarrow r_2$ denotes that the two reads are oriented toward each other (with the $5'$ ends farthest apart). Such a score can be easily shown to give higher value to layouts with as few unsatisfied constraints as possible. Note that the mate-pair score is also dependent on local coverage of the reads, so its value should be adjusted/normalized to compensate for the variation in coverage. By penalizing the score negatively and positively according to the constraints, the current formulation assumes uniform coverage. However more sophisticated score functions could be employed, if it is necessary to precisely quantify the extent to which the score varies with coverage. The mate-pair score f of the full path P is given by the sum of the scores of each pair of reads with feasible constraints in P:

$$f(P)=\sum_{r_i,r_j\in P} S_{MP}(r_i,r_j) \quad (10)$$

Note that the current formulation of S_{MP} models only mate-pairs libraries whose reads face against each other. However, most current assemblies use a mixture of paired-end and mate-pair datasets that differ in insert size and read pair orientation. SUTTA's mate-pair score can be easily adapted to support any read pair orientation and insert size.

Memory management is very important during lookahead: the subtrees are dynamically constructed and their memory deallocated as soon as the repeat boundary is resolved. Also note that the lookahead procedure is performed every time a repeat boundary is identified, so the extra work associated with the construction and scoring of the subtrees is performed only when repeated regions of the genome are assembled. Finally note that, the construction of each subtree follows the same strategy (from Algorithm 2) and uses the same overlap score (defined in section 2.3); however, recursive lookahead is not permitted. The mate-pair score introduced in (9) is used only to prune one of the two original nodes under consideration (or both, in the rare but possible scenarios, where neither of the subtrees satisfies the mate-pair constraints). This explains how the *MatePairs*() procedure from Algorithm 2 is performed.

Dead-ends and bubbles: base pair errors in short reads form next-generation sequencing produce an intuitively non-obvious set of erroneous paths in the graph and tree structures. Because perfect matching is used to compute the overlaps, according to where the base error is located two possible ambiguities need to be resolved: *dead-ends* and *bubbles*. Dead-ends consist of short branches of overlaps that extend only for very few steps and they are typically associated with base errors located close to the read ends. Bubbles instead manifest themselves as false branches that reconnect to a single path after a small number of steps. They are typically caused by single nucleotide difference carried by a small subset of reads. The lookahead procedure is easily adapted to handle these kind of structures. Specifically for dead-ends, each branch is explored up to depth W_{de} and all the branches that have shorter depth are pruned. In the case of bubbles, both branches are expanded up to depth W_{bb} and, if they converge, only the branch with higher coverage is kept and the other one pruned.

3 RESULTS

We have compared SUTTA to several well-known short read assemblers on three real datasets from Illumina next-generation sequencing data using both mated and unmated reads. The following assemblers are used in the comparison: Edena 2.1.1 (Hernandez *et al.*, 2008), Velvet 1.0.13 (Zerbino and Birney, 2008), Taipan 1.0 (Schmidt *et al.*, 2009), ABySS 1.2.3 (Simpson *et al.*, 2009), SSAKE 3.6 (Warren *et al.*, 2007) and EULER-SR 1.1.2 (Chaisson and Pevzner, 2008). Although these datasets do not represent the state of the art in sequencing technology (for example, Illumina can currently generate longer reads up to 100 bp), they have been extensively analyzed by previously published short read assemblers.

The first dataset consists of 3.86 million 35 bp unmated reads from the *Staphylococcus aureus* strain MW2 (coverage 48X, genome length 2.8 Mb). These reads are freely available from the Edena's web site (http://www.genomic.ch/edena.php). The second dataset is composed of 12.3 million 36 bp unmated Illumina reads for the *Helicobacter acinonychis* strain Sheeba genome (coverage 284X, genome length 1.5 Mb). This dataset is freely available at http://sharcgs.molgen.mpg.de/download.shtml. The third dataset consists of 20.8 million paired-end 36 bp Illumina reads from a 200 bp insert of *Escherichia coli*, strain K12 MG1655, (coverage 160X, genome length 4.6 Mb) available at the NCBI SRA (accession no. SRX000429).

The experimental results show that SUTTA has comparable performance to the best state-of-the-art assemblers based on contig size comparison. This comparison is to be interpreted in the context of our experimental evidence that the choice of the minimum overlap parameter k affects both contig size and assembly quality (presented below, see Figs 6 and 7).

3.1 Contig size analysis

Tables 1 and 2 present the comparison based on contig size analysis for all three genomes. Only contigs of minimal length 100 bp are considered in the statistics. A contig is classified as correct if it fully aligns to the genome with a minimum base similarity of 98% (for *S.aures* and *H.acininychis*) and 95% (for *E.coli*). Inspecting the results in Table 1, it is evident that SUTTA performs comparatively well relative to these assemblers. In particular, SUTTA^a, thanks to its aggressive strategy, assembles longer contigs but it pays in assembly quality by generating more misassembly errors. SUTTA^c instead behaves more conservatively and generated less errors but without excessively sacrificing contig length. The choice between the aggressive strategy and the conservative one is clearly based on the overall quality of the input set of reads and the genome structure. For example, in the case of an error-free dataset and a genome with few and short repeats, we may opt for an aggressive strategy. In the

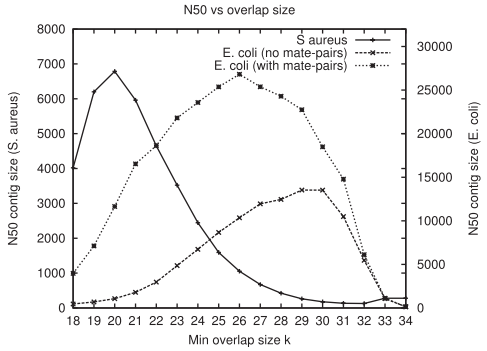

Fig. 6. Relation between the min overlap parameter k and the N50 contig size for *S.aureus* and *E.coli*.

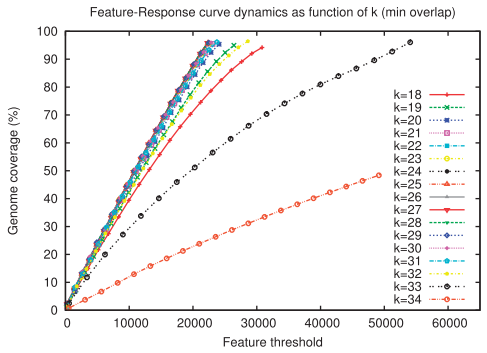

Fig. 7. Feature–response curve dynamics as a function of the minimum overlap parameter k for *E.coli* using mate-pair data.

case of mate-pair data, SUTTA produces shorter contigs compared with ABySS, Velvet and EULER-SR; however, SUTTA's overall assembly quality is superior with fewer and shorter misassembled contigs.

3.2 Overlap size k

The min overlap length k is a determinant parameter and its optimal setting strongly depends on the data (coverage). This is particularly pronounced for short reads since the required overlapping length represents a significant part of the read length. Therefore, the effective coverage $E_{cov} = \frac{N(l-k)}{G}$ is significantly sensitive to choice of the minimum overlap parameter k, where N is the number of reads, l is the read length and G is the genome size. Figure 6 shows the relation of the N50 size versus the minimum overlap parameter k for two of the genomic datasets analyzed in this article. Clearly, there is a trade-off between the number of spurious overlaps and lack of overlaps as the values for k move from small to larger numbers. Increasing the overlap allows to resolve more ambiguities, but in turn requires a higher coverage depth to achieve the same N50 value. It is important to emphasize that the optimal value for k depends on the genome structure and coverage (*S.aureus* and *E.coli* have different optimal value) and so it needs to be tuned accordingly. Finally, the availability of mate-pairs definitely improves the results and enables assembly of longer contigs for the *E.coli* genome.

Table 1. Assembly comparison for *S.aureus* (strain MW2) and *H.acininychis* (strain Sheeba)

Assembler	Number of correct	Number of errors	N50 (kb)	Mean (kb)	Max (kb)	Coverage (%)
S.aureus						
SUTTA^c	998	11	6.0	2.6	22.8	97
SUTTA^a	892	52	6.3	2.8	37.7	97
Edena (strict)	1124	0	5.9	2.4	25.7	98
Edena (non-strict)	740	16	9.0	3.7	51.8	97
Velvet	945	5	7.4	2.8	32.7	97
ABySS	928	6	7.8	2.9	32.7	98
EULER-SR	669	33	10.1	4.0	37.9	99
SSAKE	2073	378	2.0	1.1	9.7	99
Taipan	692	16	11.1	3.9	44.6	98
H.acininychis						
SUTTA^c	313	9	9.6	4.5	41.3	98
SUTTA^a	216	37	13.1	5.9	68.0	98
Edena (strict)	336	0	10.1	4.5	36.9	98
Edena (non-strict)	302	1	13.2	4.9	35.0	97
Velvet	278	2	12.8	5.4	49.5	98
ABySS	270	8	13.9	5.4	54.7	98
EULER-SR	730	21	4.3	2.1	18.8	98
SSAKE	675	156	3.2	1.8	14.6	99
Taipan	271	0	13.3	5.6	48.6	98

SUTTA^c uses the conservative approach; SUTTA^a uses the aggressive strategy.

Table 2. Assembly comparison for *E.coli* strain K12 MG1655

Assembler	Number of correct	Number of errors (mean kb)	N50 (kb)	Mean (kb)	Max (kb)	Coverage (%)
SUTTA^m	423	7 (18.8)	22.7	10.2	84.5	98
Edena	674	6 (13.2)	16.4	6.6	67.1	99
Velvet	275	9 (52.9)	54.3	15.9	166.0	98
ABySS	114	10 (49.5)	87.4	37.3	210.7	99
EULER-SR	190	26 (37.8)	57.4	21.1	174.0	99
SSAKE	407	66 (15.3)	31.2	9.6	105.9	98
Taipan	742	62 (5.2)	12.2	5.6	56.5	97

SUTTA^m uses mate-pairs constraints in the lookahead to resolve repeat boundaries.

3.3 Feature–response curve dynamics

The choice of the minimum overlap parameter k not only affects the estimated length of the assembled contigs but also changes the overall quality of the assembled sequences. In order to show this phenomenon, we use a new metric to examine the assembly quality that we call 'Feature-Response' (FR) curve. Similarly to the receiver operating characteristic (ROC) curve, the FR curve characterizes the sensitivity (e.g. coverage) of the sequence assembler as a function of its discrimination threshold (number of features/errors). Features correspond to doubtful regions of the assembled sequence and are computed using the `amosvalidate` pipeline developed by Phillippy *et al.* (2008). Faster the FR grows better is the assembly quality, because higher genome coverage is achieved with less errors (see the Supplementary Material for more details on the FR curve). Figure 7 shows the dynamics of the FR curve for *E.coli* as a function of the minimum overlap parameter k. Similarly to the plots in

Figure 6, both small and large values of k produce more assembly errors, while the best value lays in the middle range of 25–29. There seems to be a phase transition for $k=33$ and $k=34$, this is due to the fact the probability to detect a perfect match overlap of higher size ($k>32$) becomes more unlikely without increasing the coverage. Both average contig length and N50 value decrease such that more contigs of size smaller than the insert size are created. All these contigs then violate the mate-pair constraints and result in a high number of features/errors.

3.4 Computational performance

Because of the theoretical intractability of the sequence assembly problem and because, in principle, SUTTA's exploration scheme could make it generate an exponentially larger number of layouts, SUTTA could be expected to suffer from long running time and high memory requirements. However, our empirical analysis shows that SUTTA has a competitively good performance—thanks to the B&B strategy, well-defined scoring and pruning schemes, and a careful implementation. SUTTA's computational performance was compared with Velvet, ABySS, EULER-SR, Edena and SSAKE on the *S.aureus* genome using a four quad core processor machine, Opteron CPU 2.5 GHz (see the Supplementary Material for the comparative table). SUTTA has an assembly time complexity similar to Edena, SSAKE and EULER-SR. Velvet and ABySS have the best computational performance. Velvet, ABySS and Edena consume less memory than SUTTA; however, note that SUTTA relies on AMOS to maintain various genomic objects (reads, inserts, maps, overlaps, contigs, etc.), which are not optimized for short reads. At the current stage of development, SUTTA is limited to relatively small gnomes and its time complexity increases with mate-pairs constrain computation, but is expected to improve with reengineering planned for the next versions. Finally, note that typically two-thirds of total SUTTA's running time is dedicated to the computation of overlaps, leaving only a one-third of the total time to assemble the contigs.

4 CONCLUSION

Sequence assembly accuracy has now become particularly important in: (i) genome-wide association studies, (ii) detecting new polymorphisms and (iii) finding rare and *de novo* mutations. New-sequencing technologies have reduced cost and increased the throughput; however, they have sacrificed read length and accuracy by allowing more single nucleotide (base-calling) and indel (e.g. due to homo-polymer) errors. Overcoming these difficulties without paying for high computational cost requires (i) better algorithmic framework (not greedy), (ii) being able to adapt to new and combined hybrid technologies (allowing significantly large coverage and auxiliary long-rage information) and (iii) prudent experiment design.

We have presented a novel assembly algorithm, SUTTA, that has been designed to satisfy these goals as it exploits many new algorithmic ideas. Challenging the popular intuition, SUTTA enables 'fast' global optimization of the WGSA problem by taming the complexity using the B&B method. Because of the generality of the proposed approach, SUTTA has the potential to adapt to future sequencing technologies without major changes to its infrastructure: technology-dependent features can be encapsulated into the *lookahead* procedure and well-chosen *score functions*.

Funding: NSF CDI program; Abraxis BioScience, LLC.

Conflict of Interest: none declared.

REFERENCES

Applegate,D. *et al.* (2001) Tsp cuts which do not conform to the template paradigm. In *Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions [based on a Spring School]*. Springer, London, UK , pp. 261–304.

Chaisson,M.J. and Pevzner,P.A. (2008) Short read fragment assembly of bacterial genomes. *Genome Res.*, **18**, 324–330.

Davis,M. *et al.* (1962) A machine program for theorem-proving. *Commun. ACM*, **5**, 394–397.

Hernandez,D. *et al.* (2008) De novo bacterial genome sequencing: Millions of very short reads assembled on a desktop computer. *Genome Res.*, **18**, 802–809.

Karp,R.M. (2003) The role of algorithmic research in computational genomics. *Comput. Syst. Bioinformatics Conf. Int. IEEE Comput. Soc.*, **0**, 10.

Kececioglu,J. and Myers,E. (1995) Combinatorial algorithms for dna sequence assembly. *Algorithmica*, **13**, 7–51.

Kent,W.J. and Haussler,D. (2001) Assembly of the working draft of the human genome with GigAssembler. *Genome Res.*, **11**, 1541–1548.

Land,A.H. and Doig,A.G. (1960) An automatic method of solving discrete programming problems. *Econometrica*, **28**, 497–520.

Ma,B. (2009) Why greed works for shortest common superstring problem. *Theor. Comput. Sci.*, **410**, 5374–5381.

Myers,E.W. (1995) Toward simplifying and accurately formulating fragment assembly. *J. Comput. Biol.*, **2**, 275–290.

Phillippy,A. *et al.* (2008) Genome assembly forensics: finding the elusive mis-assembly. *Genome Biol.*, **9**, R55.

Sanger,F. *et al.* (1977) DNA sequencing with chain-terminating inhibitors. *Proc. Natl Acad. Sci. USA*, **74**, 5463–5467.

Schmidt,B. *et al.* (2009) A fast hybrid short read fragment assembly algorithm. *Bioinformatics*, **25**, 2279–2280.

Semple,C.A.M. (2003) Assembling a view of the human genome. In Barnes,M.R. and Gray,I.C. (eds), *Bioinformatics for Geneticists*, chapter 4. John Wiley & Sons, Ltd, Chichester, UK, pp. 93–117.

Simpson,J.T. *et al.* (2009) ABySS: a parallel assembler for short read sequence data. *Genome Res.*, **19**, 1117–1123.

Smith,T.F. and Waterman,M.S. (1981) Identification of common molecular subsequences. *J. Mol. Biol.*, **147**, 195–197.

Tarhio,J. and Ukkonen,E. (1988) A greedy approximation algorithm for constructing shortest common superstrings. *Theor. Comput. Sci.*, **57**, 131–145.

Warren,R.L. *et al.* (2007) Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, **23**, 500–501.

Zerbino,D.R. and Birney,E. (2008) Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.*, **18**, 821–829.