BIOINFORMATICS ORIGINAL PAPER

Vol. 28 no. 11 2012, pages 1429–1437
doi:10.1093/bioinformatics/bts175

Sequence analysis

Advance Access publication April 6, 2012

GRASS: a generic algorithm for scaffolding next-generation sequencing assemblies

Alexey A. Gritsenko[1,2,3,*], Jurgen F. Nijkamp[1,3], Marcel J.T. Reinders[1,2,3] and Dick de Ridder[1,2,3]

[1]The Delft Bioinformatics Lab, Department of Mediamatics, Delft University of Technology, Mekelweg 4, 2628 CD Delft, [2]Platform Green Synthetic Biology and [3]Kluyver Centre for Genomics of Industrial Fermentation, P.O. Box 5057, 2600 GA Delft, The Netherlands

Associate Editor: Michael Brudno

ABSTRACT

Motivation: The increasing availability of second-generation *high-throughput sequencing* (HTS) technologies has sparked a growing interest in *de novo* genome sequencing. This in turn has fueled the need for reliable means of obtaining high-quality draft genomes from short-read sequencing data. The millions of reads usually involved in HTS experiments are first assembled into longer fragments called *contigs*, which are then scaffolded, i.e. ordered and oriented using additional information, to produce even longer sequences called *scaffolds*. Most existing scaffolders of HTS genome assemblies are not suited for using information other than paired reads to perform scaffolding. They use this limited information to construct scaffolds, often preferring scaffold length over accuracy, when faced with the tradeoff.

Results: We present GRASS (GeneRic ASsembly Scaffolder)—a novel algorithm for scaffolding second-generation sequencing assemblies capable of using diverse information sources. GRASS offers a mixed-integer programming formulation of the contig scaffolding problem, which combines contig order, distance and orientation in a single optimization objective. The resulting optimization problem is solved using an *expectation–maximization* procedure and an unconstrained binary quadratic programming approximation of the original problem. We compared GRASS with existing HTS scaffolders using Illumina paired reads of three bacterial genomes. Our algorithm constructs a comparable number of scaffolds, but makes fewer errors. This result is further improved when additional data, in the form of related genome sequences, are used.

Availability: GRASS source code is freely available from http://code.google.com/p/tud-scaffolding/.

Contact: a.gritsenko@tudelft.nl

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

Received on March 27, 2012; revised on March 27, 2012; accepted on April 2, 2012

1 INTRODUCTION

High-throughput sequencing (HTS) technologies, such as Illumina (Illumina, Inc., San Diego, CA), 454 (Roche Applied Science, Penzberg, Germany) and SOLiD and IonTorrent (Life Technologies, Carlsbad, CA) produce millions of short DNA reads of typical lengths of 36–500 bp at low cost, making them attractive for *de novo* sequencing applications. With the aid of assembly algorithms (Miller *et al*., 2008; Peng *et al*., 2010; Zerbino and Birney, 2008), short reads can be joined together into longer sequences called *contigs*. However, contigs are typically shorter than the sequenced DNA molecules, as genomic repeat regions longer than the read length cannot be unambiguously assembled using the read sequences alone. Scaffolding, the process of using additional data to place contigs in the right order, orientation and at the right distance in longer (gapped) supercontigs called *scaffolds*, is a crucial step in obtaining high-quality draft genome sequences.

Paired reads (mate pair or paired end reads, depending on the sequencing protocol), i.e. reads of known relative orientation, order and approximate physical distance, are often used for scaffolding. Additional information, including reference sequences of related organisms, restriction maps (Nagarajan *et al*., 2008) and RNA-seq data, can be used to derive more accurate contig placement (Kent and Haussler, 2001; Pop *et al*., 2004), thereby reducing the cost of finishing experiments and allowing for more reliable downstream analyses. However, most existing scaffolding algorithms are not able to utilize such information for scaffolding. To our knowledge, only Bambus (Pop *et al*., 2004) and SOPRA (Dayarian *et al*., 2010) can make use of additional data sources, although the latter was not originally designed for this purpose.

Generally, the *contig scaffolding problem* (CSP) is finding a linear ordering and orientation of contigs that minimizes the number of unsatisfied scaffolding constraints. These constraints are derived from the available data through translation of the inherent distance, order and orientation constraints onto the contigs. The derived constraints can be mutually exclusive, which makes the problem of minimizing the number of unsatisfied constraints NP-hard (Huson *et al*., 2002; Kececioglu and Myers, 1995). Consequently, practical scaffolding algorithms only approximately solve this problem: Bambus (Pop *et al*., 2004) separately finds contig orientation and order and uses greedy heuristics to remove inconsistent constraints; SSPACE (Boetzer *et al*., 2011) greedily extends scaffolds using a heuristic stopping criterion; and SOPRA (Dayarian *et al*., 2010) uses an iterative procedure to identify a subset of contigs with consistent scaffolding constraints. Notable exceptions are OPERA (Gao *et al*., 2011) and the MIP Scaffolder (Salmela *et al*., 2011), which simplify the CSP by dropping types of constraints. OPERA implements an algorithm for finding an exact CSP solution without minimum

*To whom correspondence should be addressed.

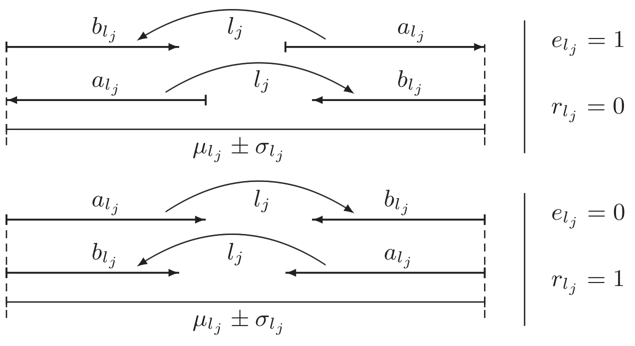

Fig. 1. Examples of contig links l_j between contigs a_{l_j} and b_{l_j} and their corresponding relative orientation (e_{l_j}), relative order (r_{l_j}) and distance ($\mu_{l_j} \pm \sigma_{l_j}$) constraints

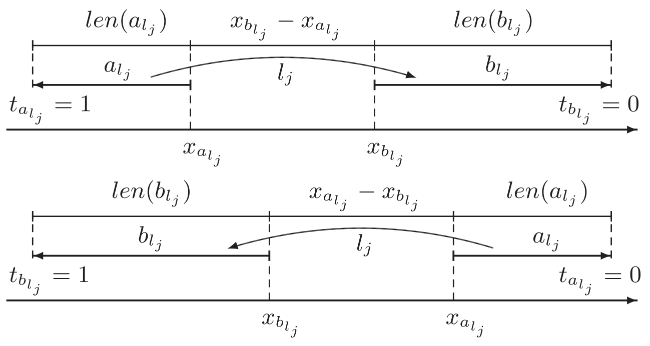

Fig. 2. Optimization variables $x_{a_{l_j}}, x_{b_{l_j}}, t_{a_{l_j}}$ and $t_{b_{l_j}}$ associated with contigs. Example for $e_{l_j} = 0 \wedge r_{l_j} = 0$

contig distance constraints; the MIP Scaffolder (Salmela *et al.*, 2011) couples a *Mixed-Integer Programming* (MIP) formulation of the CSP that does not enforce order constraints with an algorithm heuristically dividing the original problem into subproblems to be solved exactly.

We propose a novel GeneRic ASsembly Scaffolding (GRASS) algorithm that can be applied to any type of scaffolding information. Our work is similar to Salmela *et al.* (2011), as we propose a MIP formulation of the scaffolding problem. However, we combine contig orientation, order and distance in a single quadratic optimization objective. Similar to Dayarian *et al.* (2010), we employ an iterative procedure to select a consistent subset of contigs. However, we apply an expectation–maximization strategy to maximize the objective function that identifies inconsistent constraints rather than contigs, thereby retaining more scaffolding information.

We implemented the algorithm in C++ and tested it on *de novo* assemblies of paired read data for the bacteria *Eschrichia coli*, *Pseudoxanthomonas suwonensis*, and *Pseudomonas syringae* and compared it with the SSPACE, OPERA and MIP scaffolders. GRASS produces a competitive number of scaffolds with fewer scaffolding errors, particularly when combining various sources of scaffolding information.

2 METHODS

2.1 Data representation

Scaffolding constraints on contig distance, order and orientation are derived from the data in a manner depending on the data type. For example, the known *relative* orientation, *relative* order and approximate distance of paired reads that map to different contigs can be translated into relative contig orientation, order and approximate contig distance by taking mapping orientations and positions into account; similarly, physical distance, relative order and orientation of two contigs mapping to the same reference sequence can be translated into corresponding constraints. However, different data types eventually define the same type of pair-wise contig constraints, which can be conveniently represented as arcs (i.e. directed edges) $l_j = (a_{l_j}, b_{l_j}) \in E$ of weight ω_{l_j} in a digraph $G = (V, E)$ defined over the set of contigs V (Gao *et al.*, 2011; Huson *et al.*, 2002; Pop *et al.*, 2004). The weight can be chosen to reflect information source importance and consistency. A relative order r_{l_j}, relative orientation e_{l_j} and approximate distance suggested by the pair-wise constraints, are then associated with every arc l_j. The approximate distance is recorded as mean μ_{l_j} and its SD σ_{l_j}. This form is a natural choice for capturing variation in contig distances derived from the paired read insert size. It is also suitable for scaffolding constraints without (reliable) distance estimates, for example constraints derived from paired RNA-seq data of an organism with abundant intron splicing, or by mapping contigs to genome of a distant relative. Such constraints can use a large σ_{l_j} to reflect the uncertainty in the data source. We refer to l_j, its importance weight ω_{l_j}, and the corresponding contig pair-wise constraints as a *contig link*, and to G as the *contig link graph*. For succinct notation, for every contig link constraints are recorded as

- $e_{l_j} = \begin{cases} 0, & a_{l_j} \text{ and } b_{l_j} \text{ are from different strands} \\ 1, & a_{l_j} \text{ and } b_{l_j} \text{ are from the same strand} \end{cases}$
- $r_{l_j} = \begin{cases} 0, & a_{l_j} \text{ follows } b_{l_j} \\ 1, & b_{l_j} \text{ follows } a_{l_j} \end{cases}$ given that a_{l_j} has forward orientation.

This abstract definition is illustrated in Figure 1. It allows capturing any combination of contig order, distance and orientation, including constraints derived from paired end reads, mate pair reads and contig mapping.

2.2 Contig link bundling and erosion

We create a single contig link for every available piece of evidence (e.g. pair of reads) and by default set its importance weight to one (a parameter adjustable per information source). For high-coverage HTS data this procedure creates a large number of links. Contig link bundling is used to reduce the number of links, and thereby the complexity of the problem. For every ordered pair of contigs (u, v), arcs $(u, v) \in E$ that agree on contig distance, order and orientation are combined into one or more contig links as in Huson *et al.* (2002). The weight of a link after bundling is equal to the sum of weights of links bundled together to create it. Our definition of contig links permits having links that agree on all constraints, yet cannot be bundled together because they are oppositely directed in G. To enable bundling of such links, we re-set r_{l_j} relative to one of the end points of l_j to make sure that all links connecting a pair of contigs have the same directionality. Finally, contig links with importance weight smaller than a pre-defined erosion threshold e are removed from the graph. This assumes that erroneous links are rare.

2.3 Optimization formulation

We present a *mixed-integer quadratic programming* (MIQP) formulation of the CSP. Our formulation is equivalent to the traditional one [minimize the number of unsatisfied constraints, Huson *et al.* (2002)], but uses slack variables as continuous measures of the extent to which each order and orientation constraint is satisfied. This allows for uncertain data, yielding less trustworthy constraints, to be accurately exploited in the scaffolding process. A number of optimization variables are associated with every contig and contig link. We maximize an objective function f of these variables subject to scaffolding constraints expressed as linear optimization constraints. The function reaches its maximum value when all distance, order and

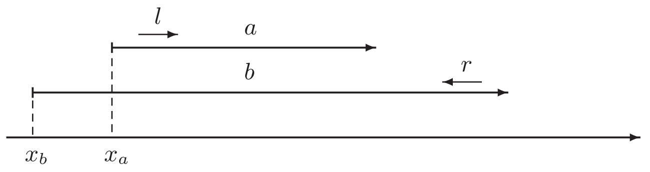

Fig. 3. Contigs *a* and *b* are not in the order predicted by mapped paired reads *l* and *r*, although the paired reads are in the correct order

orientation constraints are satisfied. Each valid collection of the optimization variable values forms a solution to the optimization problem. These values are sufficient to puzzle contigs into scaffolds. For every contig c_i, $i=1,\dots,n$ the following variables are defined, as illustrated in Figure 2.

- $t_i = \begin{cases} 0, & c_i \text{ comes from the forward strand of the scaffold} \\ 1, & c_i \text{ comes from the reverse strand of the scaffold} \end{cases}$
  is used to define contig orientation in the scaffold.

- $x_i \in \mathbb{R}^+$ corresponds to the 5′ position of c_i in the scaffold (when input contigs and the constructed scaffold are viewed as having a 5′ to 3′ orientation).

Naturally x_i should be an integer variable, but it is relaxed to simplify the optimization problem and is rounded to the nearest integer when the solution is converted into scaffold nucleotide sequences. Additionally, with every link $l_j, j=1,\dots,m$ the following variables are associated:

- **Slack variables** for distance constraints, $\xi_{l_j} = \{\overrightarrow{\xi_{l_j}}, \overleftarrow{\xi_{l_j}}\} \in \mathbb{R}^+ \times \mathbb{R}^+$, and order constraints, $\Delta_{l_j} = \{\overrightarrow{\Delta_{l_j}}, \overleftarrow{\Delta_{l_j}}\} \in \mathbb{R}^+ \times \mathbb{R}^+$, for forward $(t_{a_{l_j}} = 0)$ and reverse $(t_{a_{l_j}} = 1)$ orientations of the contig pair, respectively. By design these variables reflect the degree to which the corresponding constraints are violated. They are penalized in the optimization objective f.

- **Switch variables** for distance constraints, $\alpha_{l_j} \in \{0,1\}$, and order constraints, $\beta_{l_j} \in \{0,1\}$ (0, constraint is disabled; 1, enabled) used for disabling contig link constraints with high penalties.

As distance and order constraints are influenced by the orientation, different slack variables are required for both orientations. We omit orientation arrows above slacks ξ_{l_j} and Δ_{l_j} when the contig pair orientation is not important, or is clear from the context.

Contig links impose scaffolding constraints, which can be modeled as MIQP optimization constraints. We demonstrate here how such constraints can be derived from paired read data; the same type of constraints can be derived in a similar way from other sources of scaffolding information (for example, see Section 3.2).

Distance constraints are expressed as:

$$\frac{|d(a_{l_j}, b_{l_j}) - \mu_{l_j}|}{\sigma_{l_j}} \le \xi_{l_j}, \tag{1}$$

where $d(a_{l_j}, b_{l_j})$ is the distance between contigs a_{l_j} and b_{l_j}, and ξ_{l_j} is a distance slack variable. This inequality captures uncertainty in the distance by measuring the difference with the mean in SDs. We derive contig distance $d(a_{l_j}, b_{l_j})$ from the paired read insert size as the gap size plus the contig lengths. The calculation then depends on the order and orientation of contigs connected by l_j. It can be fixed by assuming that the contigs have relative orientation and order suggested by l_j. For example, for the case of $(e_{l_j} = 0 \wedge r_{l_j} = 0)$ shown in Figure 2, the distance expression depends on contig pair orientation through $t_{a_{l_j}}$:

$$d(a_{l_j}, b_{l_j}) = x_{a_{l_j}} - x_{b_{l_j}} + \mathrm{len}(a_{l_j}) + \mathrm{len}(b_{l_j}),\ t_{a_{l_j}} = 0$$
$$d(a_{l_j}, b_{l_j}) = x_{b_{l_j}} - x_{a_{l_j}} + \mathrm{len}(a_{l_j}) + \mathrm{len}(b_{l_j}),\ t_{a_{l_j}} = 1.$$

Combined with (1) the following constraints are obtained:

$$\begin{cases} x_{a_{l_j}} - x_{b_{l_j}} \le \sigma_{l_j}\overrightarrow{\xi_{l_j}} + \mu_{l_j} - \mathrm{len}(a_{l_j}) - \mathrm{len}(b_{l_j}) \\ x_{a_{l_j}} - x_{b_{l_j}} \ge -\sigma_{l_j}\overrightarrow{\xi_{l_j}} + \mu_{l_j} - \mathrm{len}(a_{l_j}) - \mathrm{len}(b_{l_j}) \\ x_{b_{l_j}} - x_{a_{l_j}} \le \sigma_{l_j}\overleftarrow{\xi_{l_j}} + \mu_{l_j} - \mathrm{len}(a_{l_j}) - \mathrm{len}(b_{l_j}) \\ x_{b_{l_j}} - x_{a_{l_j}} \ge -\sigma_{l_j}\overleftarrow{\xi_{l_j}} + \mu_{l_j} - \mathrm{len}(a_{l_j}) - \mathrm{len}(b_{l_j}) \end{cases}, \tag{2}$$

where different slack variables are used for the two contig pair orientations. The expressions for other combinations of e_{l_j} and r_{l_j} are derived similarly.

Order constraints are derived from read order constraints (i.e. if c_j follows c_i, then they should not overlap and c_j must be upstream of c_i), which additionally can be relaxed. The relaxation is necessary because (i) assembled contigs may overlap (Pop *et al.*, 2004); (ii) in some cases the order constraints on data are not valid when extended to contigs, as illustrated in Figure 3. Translating order constraints into optimization constraints as

$$\begin{cases} x_{a_{l_j}} - x_{b_{l_j}} \ge -\mathrm{len}(b_{l_j}) \cdot \overrightarrow{\Delta_{l_j}},\ t_{a_{l_j}} = 0 \\ x_{b_{l_j}} - x_{a_{l_j}} \ge -\mathrm{len}(a_{l_j}) \cdot \overleftarrow{\Delta_{l_j}},\ t_{a_{l_j}} = 1 \end{cases} \tag{3}$$

(formulas shown for $e_{l_j} = 0 \wedge r_{l_j} = 0$) discourages overlaps while still allowing the order constraint to be violated when $\Delta_{l_j} > 1$. These slack variables are weighed by the length of the downstream contig to allow measuring them on a single scale. As for the distance optimization constraints, it is assumed that the relative contig orientation is correct.

Orientation constraints are modeled in the optimization objective function, which is designed to attain larger values when more orientation constraints are satisfied. The function is given by a polynomial

$$g(t) = \sum_{j=1,\dots,m}^{e_{l_j}=0} q_{a_{l_j} b_{l_j}} \omega_{l_j} + \sum_{j=1,\dots,m}^{e_{l_j}=1} (1 - q_{a_{l_j} b_{l_j}}) \omega_{l_j},$$

where $q_{ab} = t_a + t_b - 2t_a t_b \equiv \begin{cases} 0, & a \text{ and } b \text{ are equally oriented} \\ 1, & \text{otherwise} \end{cases}$.
It is equal to the sum of weights of contig links with satisfied orientation and serves as a basis for the optimization objective that is further penalized proportionally to slack variables.

Slack penalties: the distance and order constraints are added to the optimization problem through slack variable penalization. The penalty is proportional to the importance weight of the corresponding contig link and to the value of the slack variable. To avoid situations when a low-weight violated constraint results in a large penalty, a maximum penalty of half of the importance weight is enforced, after which the constraint is considered disabled. Doing this has the additional benefit of equalizing the influence of order and distance constraints. To this end we penalize as follows

$$\frac{\omega_{l_j}}{2} \cdot \frac{\min(\xi_{l_j}, S_\xi)}{S_\xi}, \tag{4}$$

where ξ_{l_j} is chosen as $\overrightarrow{\xi_{l_j}}$ or $\overleftarrow{\xi_{l_j}}$, according to the contig pair orientation and S_ξ is the maximum slack threshold (after which the slack is disabled). Because the expression $\min(\xi, S_\xi)$ is not suitable for direct use in a MIP, it is unrolled using the switch variables as $[\alpha_{l_j}\xi_{l_j} + (1-\alpha_{l_j})S_\xi]$. Similar penalties with variables Δ_{l_j} and β_{l_j}, and maximum slack threshold S_Δ are used for the order constraints. We set $S_\xi = 6$ (i.e. six SDs) as in Gao *et al.* (2011); Li and Durbin (2009); and $S_\Delta = 1$, as at this value of slack the physical order constraint is not satisfied anymore. Further, only the slacks for the appropriate contig pair orientation have to be penalized. This is achieved by penalizing $(1-t_{a_{l_j}})\overrightarrow{\xi_{l_j}} + t_{a_{l_j}}\overleftarrow{\xi_{l_j}}$ in place of ξ_{l_j} in (4). This expression 'chooses' which slack variable to penalize depending on the contig pair orientation. Finally, the constraints have to be penalized only when they are meaningful (i.e. the relative contig orientation e_{l_j} is assumed to be satisfied). The resulting function looks as follows:

$$h(t,\alpha,\xi,S_\xi)=\sum_{j=1,\ldots,m}^{e_{l_j}=0} q_{a_{l_j}b_{l_j}}\frac{\omega_{l_j}}{2S_\xi}\left[(1-t_{a_{l_j}})\overrightarrow{\xi}_{l_j}+t_{a_{l_j}}\overleftarrow{\xi}_{l_j}\right]+$$

$$+\sum_{j=1,\ldots,m}^{e_{l_j}=1}(1-q_{a_{l_j}b_{l_j}})\frac{\omega_{l_j}}{2S_\xi}\left[(1-t_{a_{l_j}})\overrightarrow{\xi}_{l_j}+t_{a_{l_j}}\overleftarrow{\xi}_{l_j}\right].$$

Expansion of this function leads to a fourth degree polynomial, containing only terms that consist purely of binary variables, or one continuous and up to three binary variables. To construct a MIQP formulation, using the big-M formulation (Nemhauser and Wolsey, 1988), these terms can be replaced by a single new auxiliary variable each at the expense of introducing new optimization constraints.

Putting it all together: we maximize

$$f(x,t,\alpha,\beta,\xi,\Delta)\equiv g(t)-h(t,\alpha,\xi,S_\xi)-h(t,\beta,\Delta,S_\Delta),$$

s.t. constraints (2) and (3) are satisfied. Here $g(t)$ is maximized for orientation, $h(t,\alpha,\xi,S_\xi)$ is minimized for orientation and distance, and $h(t,\beta,\Delta,S_\Delta)$ is minimized for orientation and order, in a single optimization objective. Given the NP-hard nature of MIPs and the large number of binary variables in the proposed formulation, this problem becomes intractable even for small numbers of contigs.

2.4 Problem splitting

We tackle this intractability with an *expectation–maximization* (EM) like procedure.

The maximization step assumes the contig orientations are known (i.e. t_i and q_{ab} are fixed). Knowing t_i allows us to choose the slack variables ($\overrightarrow{\xi}_{l_j}$ or $\overleftarrow{\xi}_{l_j}$, and $\overrightarrow{\Delta}_{l_j}$ or $\overleftarrow{\Delta}_{l_j}$) depending on the contig pair orientations, and to select contig links with satisfied relative orientation before the optimization problem is constructed, significantly reducing the number of optimization constraints and the complexity of the optimization problem:

$$\begin{aligned}&f(x,\alpha,\beta,\xi,\Delta)=g-h(\alpha,\xi,S_\xi)-h(\beta,\Delta,S_\Delta)\\&g=\sum\omega_{l_j}\equiv\text{const},\quad h(\alpha,\xi,S_\xi)=\sum\min(\xi_{l_j},S_\xi)\cdot\frac{\omega_{l_j}}{2S_\xi}\end{aligned}\quad .\qquad (5)$$

This *fixed optimization problem*, however, is still NP-hard due to the binary variables α_{l_j} and β_{l_j} involved in expansion of the min terms. We obtain an approximate solution to this problem by first exactly solving its continuous relaxation, choosing α_{l_j} and β_{l_j} according to the slack values in the relaxation solution and finally, re-solving the problem with these values fixed. The relaxation is obtained by replacing $h(\alpha,\xi,S_\xi)$ by $h(\xi,S_\xi)=1/2S_\xi\sum\omega_{l_j}\xi_{l_j}$ in (5). This eliminates all binary variables, allowing the use of efficient optimization algorithms (Dantzig, 1988). The solution for the relaxed problem gives us optimal values for slacks ξ_{l_j} and Δ_{l_j}, which are used to choose α_{l_j} and β_{l_j} as

$$\alpha_{l_j}=\begin{cases}0, & \xi_{l_j}>S_\xi\\ 1, & \xi_{l_j}\le S_\xi\end{cases},\quad \beta_{l_j}=\begin{cases}0, & \Delta_{l_j}>S_\Delta\\ 1, & \Delta_{l_j}\le S_\Delta\end{cases},$$

and allows us to re-solve problem (5). The rationale behind is that, since the majority of link information is assumed to be correct, large slack values will be associated with incorrect constraints that have to be disabled. The total penalty for l_j is memorized (initially set to zero) for use in the expectation step as

$$\Theta_{l_j}\leftarrow\frac{\min(\xi_{l_j},S_\xi)}{2S_\xi}\omega_{l_j}+\frac{\min(\Delta_{l_j},S_\Delta)}{2S_\Delta}\omega_{l_j}.$$

The expectation step is used to obtain the expected contig orientations t_i, which maximize the objective function for the *previously observed* penalties. Consider the MIQP problem when penalties associated with the links are known (i.e. Δ_{l_j}, ξ_{l_j}, α_{l_j} and β_{l_j} are fixed), and the optimal contig orientation is sought. In this problem, when a contig link is enabled, its weight is penalized by the associated slack Θ_{l_j}. We can, therefore, consider an equivalent problem where all slacks are zero and link weights are modified as $\tilde{\omega}_{l_j}\leftarrow\omega_{l_j}-\Theta_{l_j}$. The problem is then to maximize

$$f(t)\equiv g(t)=\sum_{j=1,\ldots,m}^{e_{l_j}=0}q_{a_{l_j}b_{l_j}}\tilde{\omega}_{l_j}+\sum_{j=1,\ldots,m}^{e_{l_j}=1}(1-q_{a_{l_j}b_{l_j}})\tilde{\omega}_{l_j}\qquad (6)$$

free of any constraints. This is an *unconstrained binary quadratic programming* (UBQP) problem (Beasley, 1998), the problem of maximizing a function $c(t)=t^tCt$, where x is a binary vector of length n and C is an $n\times n$ real matrix. Consider a vector of orientations $t\in\{0,1\}^n$ and a matrix C of size n. Starting from a zero matrix, $C=(c_{ij})$ can be obtained by updating it for every link $l_j=(a,b)$ as

$$c_{aa}\leftarrow(-1)^{e_{l_j}}\tilde{\omega}_{l_j}+c_{aa},\qquad c_{bb}\leftarrow(-1)^{e_{l_j}}\tilde{\omega}_{l_j}+c_{bb}$$

$$c_{ab}\leftarrow(-1)^{e_{l_j}+1}\cdot 2\tilde{\omega}_{l_j}+c_{ab}.$$

The functions $f(t)$ and $c(t)$ will then differ by a constant and, therefore, reach maxima for the same t. Solving a UBQP is known to be an NP-hard, but well-studied problem with efficient heuristic algorithms available (Merz and Katayama, 2004; Nesterov, 1997; Pardalos *et al*., 2008). Thus, the UBQP formulation of the problem is preferred over (6) for obtaining values of t_i.

The EM steps are iterated while contig orientations change. The algorithm can be viewed as an iterative UBQP approximation of the original MIQP problem. In practice, it converges to a solution within seven iterations.

2.5 Scaffold extraction and post-processing

Repeat contigs in the contig link graph G are connected by ambiguous links, hindering a confident positioning in scaffolds. In a pre-processing step, we detect such contigs using a modification of the A-statistic (Myers *et al*., 2000) proposed by Zerbino (2009), and prevent their incorporation in scaffolds by removing all links from G incident to them. The connected components of G correspond to separate subproblems, which are solved independently.

After optimization, each solution tuple (x,t,α,β) and corresponding subgraph G' are converted into one or more scaffolds. First, contig links with disabled constraints (i.e. $\alpha_{l_j}=0\vee\beta_{l_j}=0$) are removed from G' to minimize the chance of incorrectly incorporating contigs in the same scaffold. Every connected component of the resulting G' is used to construct a single nucleotide sequence. Contigs are processed in order of their downstream end coordinates. The left end of the first contig is put at the start of the sequence; every new contig is added to the scaffold such that the gap between two consecutive contigs is preserved. When consecutive contigs are predicted to overlap (i.e. have a negative gap size), the new contig is pushed upstream to eliminate the overlap.

Because resolving contig overlaps in this way potentially leads to erroneous sequence reconstruction, we also explore an optional post-processing approach that performs global sequence alignment on consecutive contigs to find the best overlap. Global alignment is performed using a divide-and-conquer version of the Needleman–Wunsch algorithm (Hirschberg, 1975). Algorithm implementation from the NCBI C++ Toolkit was used (National Center for Biotechnology Information, 2011). For every consecutive pair of contigs predicted to have a gap of μ bp, all gap sizes of at most $d=100$ bp away from the predicted value are examined. Negative gap sizes indicate overlaps. For each gap size g, global alignment of overlapping contig ends is performed (match score of $p_{\text{match}}=2$, mismatch penalty of $p_{\text{mismatch}}=-3$). The best gap size is then chosen based on the alignment score S and proximity to the predicted gap size μ by maximizing

$$\frac{S}{g\cdot p_{\text{match}}}\cdot\frac{d-|g-\mu|}{d}.\qquad (7)$$

With the (mis)match scores chosen as above, this expression takes values in $[-1.5;1]$. Due to computational complexity only overlaps of no more than 1500 bp are considered (gap sizes with longer overlaps are assigned a score of -1). The decision to join two contigs, to leave a gap between them or to split the scaffold is then made:

- If none of the considered gap sizes suggest overlaps, the two contigs are positioned in a scaffold with a gap of μ bp.
- If value of expression (7) for the chosen gap size g passes a quality threshold of 0.8, the contigs are positioned to have an overlap of g bp. The overlap is replaced with the alignment consensus sequence, where mismatches are masked with unknown nucleotides.
- If the chosen gap size does not pass the quality threshold and is shorter than 50 bp, the two contigs are positioned successively one following another with no overlap.
- Finally, if the chosen gap size suggests a longer overlap, the currently constructed scaffold is split into two with a new scaffold starting from a contig that was predicted to lie upstream.

In principle, the proposed post-processing step with scaffold splitting allows for construction of more accurate scaffolds compared with the naïve scaffold extraction. We refer to the combination of GRASS and post-processing as GRASS+.

2.6 Evaluation criteria

Similar to assemblies, scaffolds are evaluated based on accuracy and contiguity. Scaffold accuracy can be assessed by comparing scaffolds to available reference sequences. We adopted the evaluation criteria from Dayarian *et al*. (2010); Gao *et al*. (2011) and counted the number of scaffold breakpoints, i.e. consecutive contig pairs in the scaffold that do not agree with the reference on contig distance, order or orientation. We perform local alignment of scaffolds to the reference and count the number of breakpoints within each scaffold. Two consecutive alignments are counted as a breakpoint if any of these hold: (i) they align to two different chromosomes in the reference; (ii) their relative orientations in the scaffold and in the reference do not match; (iii) their relative orders in the scaffold and in the reference do not match; (iv) the difference in distance in the scaffold and in the reference is larger than Δ. We used $\Delta = 10$ kb and $\Delta = 500$ bp to asses contig distance correctness at low and high resolution, respectively.

MUMmer (Delcher *et al*., 2002) was used to align scaffolds to references. Best hits for each position in the scaffold were computed. Only hits with at least 90 aligned bases (alignment length $\times$ alignment identity), were taken into account. In practice, very few alignments do not pass this cutoff. The alignments are also used to calculate the percentage of the scaffold bases and the reference bases that are aligned (Salmela *et al*., 2011). These numbers capture scaffold accuracy and completeness.

Finally, scaffold completeness and contiguity are captured as in sequence assembly, calculating total length of all scaffolds, number of scaffolds, maximum scaffold length and the N50 statistic.

3 IMPLEMENTATION

GRASS source code is available under the GNU GPL v3 license. It was developed in C++ and tested on Linux. GRASS consists of *linker* and *scaffolder* modules. The linker takes contigs and the available information sources as input and produces linking and coverage data, which is then used by the scaffolder module. It filters out repeat contigs and uses the remaining data to produce scaffolds. Scaffolds are output both as lists of contigs with assigned coordinates and orientations, and as linear FASTA sequences with gaps.

3.1 Paired read data processing

To obtain contig links from paired read data, the linker module performs single-end mapping of the reads to contigs. The algorithm used for mapping depends on the data type: BWA (Li and Durbin, 2009) for Illumina reads, NovoAlign (http://www.novocraft.com/) for 454 data. The aligners are set to output all mapping locations, including non-unique hits, as a SAM file (Li *et al*., 2009), which is then converted to BAM for further processing. This process is applied to each paired read library.

Read alignments are pre-processed to remove read pairs with low quality and ambiguous alignments. As a rule, only unique hits with no mismatches and minimum read length of 30 bp are kept. The filtered alignments are then scanned for paired reads that align to different contigs. Each such read pair mapping is used to create a single contig link with distance, order and orientation constraints derived from the mapping and the given read pairing method (i.e. paired ends or mate pairs). The BamTools API (Barnett *et al*., 2011) is used for filtering and processing read alignments.

3.2 Related genome data processing

An available reference sequence, such as the genome of a related organism, can be used for guiding the scaffolding process. For this purpose, contigs are aligned to the reference sequence. For every contig, a position in the reference sequence is obtained from contig tiling constructed from local alignments using MUMmer. Contig links are then created for every pair of consecutive contigs aligning to the same reference sequence, with relative orientation and order derived from the tiling. To capture alignment quality, weights for links $l_j = (a_{l_j}, b_{l_j})$ are set to $I_{a_{l_j}} \times I_{b_{l_j}} \times C_{a_{l_j}} \times C_{b_{l_j}} \times W$, where $I_{a_{l_j}}$ and $I_{b_{l_j}}$ are alignment identities, $C_{a_{l_j}}$ and $C_{b_{l_j}}$ are alignment coverages reported by MUMmer for the corresponding contigs, and $W > 0$ is a weight assigned to the reference sequences as a scaffolding information source. This procedure is applied for each available reference sequence to create links, which are then used together in the optimization.

3.3 Optimization problem solution

The EM procedure proposed for solving the MIQP formulation of the CSP splits it into a continuous *linear programming* (LP) problem, and an UBQP problem. Although more efficient algorithms for solving UBQPs are available (Nesterov, 1997; Pardalos *et al*., 2008), a memetic algorithm from Merz and Katayama (2004) was chosen for ease of implementation. Usually, contig link graphs are sparse due to the linear scaffold structure that they encompass. Memetic algorithms improve individual solutions through local search, which in turn is well-suited for smooth search landscapes (as in the case of sparse contig link graphs). Graph sparsity is further exploited by implementing sparse matrix operations as in Merz and Katayama (2004).

We use the C++ Concert API for the CPLEX Optimizer (IBM ILOG, 2011) to solve LPs. CPLEX is freely available for academic use.

4 RESULTS AND DISCUSSION

4.1 Experimental setup

We have evaluated GRASS on *de novo* HTS assemblies of three bacterial genomes: *E.coli* K12, substr. MG1655; *P.suwonensis* 11-1; and *P.syringae* B728a. For these organisms, finished genome sequences and HTS data from resequencing experiments are available. Presence of a finished genome sequence allows for reliably evaluating the algorithm and comparing it to other scaffolders in a *de novo* setup. This is achieved by using the reference sequence only in scaffold evaluation (thus not as an additional information source in the scaffolding process). The available test data is summarized in Table 1. Insert size and coverage were obtained from paired read mapping using BWA and BEDTools (Quinlan and Hall, 2010).

Velvet (Zerbino and Birney, 2008) was used to assemble reads into contigs. All assemblies had a coverage cutoff of 6 and were not scaffolded by the assembler. Only contigs longer than 150 bp were kept. Repeat resolution was disabled (i.e. no expected coverage was provided). For each organism, the k-mer length was chosen

Table 1. Available datasets NCBI/EBI accession numbers are given for reference sequences and read sets. In all cases reads were produced by the Illumina sequencing platform.

	E.coli		*P.suwonensis*		*P.syringae*
Genome size	4.64 Mb		3.42 Mb		6.09 Mb
Reference	NC_000913.2		CP002446.1		NC_007005.1
Dataset	SRR001665	SRR001666	SRR097515	SRR191848	ERR005143
Read count	$2\times 10\,408\,224$	$2\times 7\,047\,668$	$2\times 23\,960\,004$	$2\times 19\,789\,425$	$2\times 3\,551\,133$
Read length	36 bp	36 bp	76 bp	76 bp	36 bp
Coverage	$160\times$	$107\times$	$709\times$	$824\times$	$38\times$
Insert size	216 ± 10	488 ± 18	189 ± 17	189 ± 17	401 ± 33

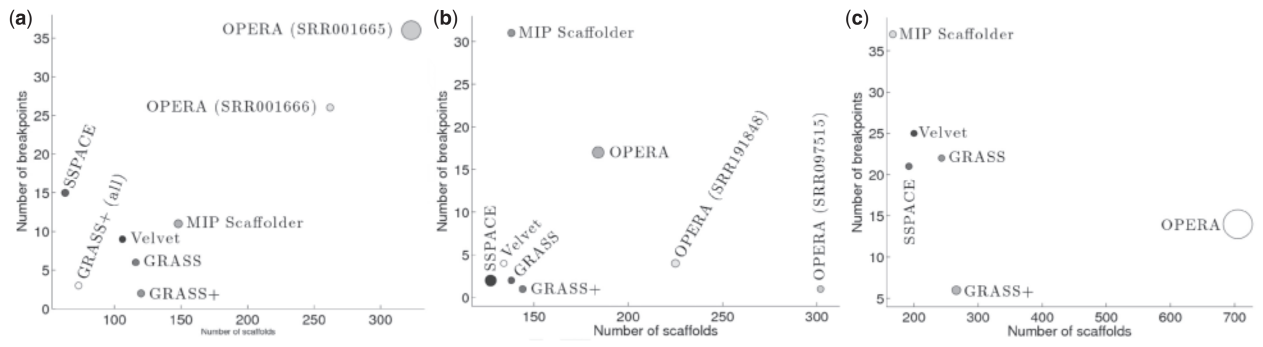

Fig. 4. Scaffold accuracy and contiguity tradeoff on the available datasets. Marker size indicates scaffolding running time in minutes, exact numbers are given in Table S4. GRASS+ using paired reads and two related genomes is shown in (a) as 'GRASS+ (all)'. (**a**) *E.coli* data. Running time ranges in [8 s; 27 min 49s]. (**b**) *P.suwonensis* data. Running time ranges in [13 s; 8 min 19s]. (**c**) *P.syringae* data. Running time ranges in [1 s; 72 min 22 s]

by performing assemblies for various k and choosing one based on assembly contiguity, length, percentage of mapped single reads, and percentage of properly paired reads (Li and Durbin, 2009) (Supplementary Tables S1, S2 and S3). For *E.coli*, *P.suwonensis* and *P.syringae*, $k=31$, $k=59$ and $k=23$ were chosen, respectively. This way of choosing k reflects real-life *de novo* assembly scenarios, yielding a realistic algorithm evaluation. Final assemblies are characterized in Tables 2–4.

4.2 Comparison to other scaffolders

We compared GRASS with SSPACE, MIP and OPERA scaffolders. Where required, insert size estimates from Table 1 were used. Tables 2–4 show evaluation metrics calculated for these scaffolders and the available test data. Unless stated otherwise, all scaffolders were run with default parameter settings. BWA was used to map reads to scaffolds and produces SAM files required by the MIP Scaffolder. As in Salmela *et al.* (2011), at most two mismatches were allowed in read mapping. For SSPACE and OPERA, reads were aligned with Bowtie (Langmead *et al.*, 2009) using scripts provided with the scaffolders.

GRASS used an erosion cutoff of 4 (although better results can be obtained by tuning this parameter) and coverage estimates obtained from exact mapping of the reads to the assembly contigs. The latter is available from output of the linker module.

The SSPACE maximum distance parameter was set to 6 SDs for each paired library. Libraries were input in order of increasing insert size.

The MIP Scaffolder was also provided with coverage estimates computed from exact read mapping. Following the original publication, we tried different filtering parameters (ω,p) and chose those which gave the highest N50 value. Settings $(36,0.8)$, $(70,0.4)$ and $(50,0.6)$ were selected for the *E.coli*, *P.suwonensis* and *P.syringae* data, respectively. Maximum partition sizes were set to 100 for the *E.coli* scaffolds and 50 for the *P.suwonensis* and *P.syringae* scaffolds. Maximum and minimum insert sizes were chosen by adding and subtracting 6 SDs to the mean insert size.

OPERA does not allow using multiple read sets. It was applied to each read library separately, and in the case of *P.suwonensis*, also to a join of the available read sets, as they have the same insert size. The minimum contig length was set to 150 bp, i.e. the contig length cutoff parameter used in Velvet. We used the default PET parameter value whenever possible and increased it to the minimum value that allowed OPERA to finish without triggering a timeout abort. Cutoff values 6 and 7 were used for the *E.coli* dataset; cutoffs 27, 5 and 5 were used for the *P.suwonensis* dataset; and 11 was used for the *P.syringae* dataset (values are given in the order of the experiments in Tables 2–4).

SOPRA was applied to assembly graphs produced by Velvet. However, when used with parameters chosen in accordance to the manual provided, SOPRA produced highly fragmented scaffolds compared with results from Salmela *et al.* (2011). To allow for a fair comparison, its results were not taken into account.

Table 2. Contiguity and accuracy statistics of the initial assembly of *E.coli* and its scaffolds

Scaffolder	Breakpoints $\Delta = 10$ kb	Breakpoints $\Delta = 500$ bp	Number of scaffolds	N50	Maximum length (bp)	Total length (bp)	Reference covered (%)	Scaffolds covered (%)
Velvet contigs	1	1	481	19 872	73 062	4 535 181	97.44	99.79
Velvet scaffolds	9	10	106	171 726	312 219	4 561 490	97.98	99.74
SSPACE	15	16	**63**	178 023	374 265	4 547 685	97.79	99.70
GRASS	6	6	116	117 964	267 989	4 546 975	97.53	99.55
GRASS+	**2**	**2**	120	112 254	268 030	4 546 640	97.53	99.55
MIP Scaffolder	11	12	148	89 070	221 548	4 546 430	97.54	99.59
OPERA (SRR001665)	36	38	323	32 799	131 842	4 544 447	97.52	99.67
OPERA (SRR001666)	26	28	262	37 330	126 797	4 556 203	97.52	99.42

Results with the smallest number of breakpoints or scaffolds are shown in bold.

Table 3. Contiguity and accuracy statistics of the initial assembly of *P.suwonensis* and its scaffolds

Scaffolder	Breakpoints $\Delta = 10$ kb	Breakpoints $\Delta = 500$ bp	Number of scaffolds	N50	Maximum length (bp)	Total length (bp)	Reference covered (%)	Scaffolds covered (%)
Velvet contigs	1	1	303	26 043	90 572	3 394 128	99.01	99.90
Velvet scaffolds	4	5	134	57 614	153 169	3 395 148	99.03	99.78
SSPACE	2	2	**127**	60 526	151 961	3 388 872	99.09	99.99
GRASS	2	2	138	62 908	152 258	3 394 155	99.02	99.91
GRASS+	**1**	**1**	144	53 211	151 938	3 389 098	99.02	99.91
MIP Scaffolder	31	32	138	52 743	115 278	3 390 104	99.03	99.93
OPERA	17	18	184	45 559	186 349	3 413 751	99.01	99.34
OPERA (SRR097515)	**1**	**1**	302	26 053	90 582	3 397 028	99.01	99.81
OPERA (SRR191848)	4	4	225	34 214	90 582	3 397 065	99.02	99.84

Results with the smallest number of breakpoints or scaffolds are shown in bold.

As a scaffolder, Velvet was provided with mean insert size and SD for each library. The data was reassembled with repeat resolution (expected coverage estimated automatically) and scaffolding turned on. Its performance was used as a baseline over which all scaffolders improved on *P.syringae* data and only SSPACE and GRASS improved on *E.coli* and *P.suwonensis* data.

Tables 2–4 contain the results. Note that the minimum number of breakpoints is one, due to the circular structure of bacterial genome. Breakpoints at $\Delta = 10$ kb and $\Delta = 500$ bp differ only slightly, suggesting that gap lengths are estimated with high precision. SSPACE produced the longest scaffolds for *E.coli*. It also produced the smallest number of scaffolds for *E.coli* and *P.suwonensis*. The longest scaffolds and the smallest number of scaffolds on the *P.syringae* dataset are achieved by the MIP Scaffolder. Similar scaffold and reference coverage percentages were achieved by all scaffolders. However, GRASS+ has the smallest number of breakpoints for all considered organisms. Additionally, for the case of *P.suwonensis*, GRASS constructed the longest scaffolds and GRASS+ produced breakpoint-free scaffolds while providing a 2-fold reduction in the number of contigs. Scaffolds produced by the MIP Scaffolder and OPERA are either very fragmented or have a large number of breakpoints.

When constructing scaffolds, scaffolding algorithms balance between scaffold contiguity and scaffold accuracy. This tradeoff is captured in Figure 4 by plotting the number of breakpoints (at $\Delta = 10$ kb) against the number of scaffolds. A good scaffolder would be located in the lower left corner of such a plot. In many cases, GRASS combines a smaller number of breakpoints with a small number of scaffolds, compared with other scaffolders. The MIP Scaffolder and SSPACE can achieve smaller numbers of scaffolds, but at the cost of (much) larger numbers of breakpoints. Clearly, GRASS and SSPACE represent two possible choices of scaffolding algorithms, with GRASS being more accurate with respect to the number of breakpoints and SSPACE constructing longer scaffolds. This behavior of the two algorithms is consistent over all datasets.

We also measured scaffolding running times, these are depicted in Figure 4 using marker size. Exact numbers, as well as read mapping running times are available in Table S4. Like most scaffolders, GRASS spends a majority of its time on read alignment, making running times of different scaffolders comparable and running time of the core scaffolding part of GRASS on the considered datasets negligible. Based on simulation results, we do not expect computation to become a bottleneck for large genomes. Nevertheless, to reduce computational load it is always possible to

Table 4. Contiguity and accuracy statistics of the initial assembly of *P.syringae* and its scaffolds

Scaffolder	Breakpoints $\Delta = 10$ kb	Breakpoints $\Delta = 500$ bp	Number of scaffolds	N50	Maximum length (bp)	Total length (bp)	Reference covered (%)	Scaffolds covered (%)
Velvet contigs	1	1	1560	8599	46 055	5 902 217	96.41	99.78
Velvet scaffolds	25	27	200	122 286	683 615	6 012 535	97.78	99.37
SSPACE	21	26	192	87 996	520 403	5 946 936	96.61	99.09
GRASS	22	25	243	85 493	618 916	5 931 679	96.57	99.38
GRASS+	**6**	**7**	266	77 945	460 726	5 945 096	96.56	99.01
MIP Scaffolder	37	47	**167**	94 327	279 875	5 943 358	96.58	99.17
OPERA	14	14	705	18 108	76 357	5 950 236	96.58	99.17

Results with the smallest number of breakpoints or scaffolds are shown in bold.

Table 5. Contiguity and accuracy statistics of *E.coli* scaffolds obtained with GRASS+ using additional data

Reads used	DH10B used	BW2952 used	Breakpoints $\Delta = 10$ kb	Breakpoints $\Delta = 500$ bp	Number of scaffolds	N50	Maximum length (bp)	Total length (bp)	Reference covered (%)	Scaffolds covered (%)
Yes	No	No	**2**	**2**	120	112 254	268 030	4 546 640	97.53	99.55
No	Yes	No	10	66	105	425 724	1 948 314	5 047 825	97.51	89.65
	No	Yes	4	65	90	843 564	1 099 102	4 773 879	97.52	92.64
	Yes	Yes	6	70	81	612 889	1 315 367	4 763 935	97.52	94.80
Yes	Yes	No	3–6	40–49	72–120	≈ 273 503	≈ 850 450	≈ 4 804 124	≈ 97.52	≈ 94.99
	No	Yes	2–7	51–55	**67–80**	≈ 497 383	≈ 1 077 789	≈ 4 569 001	97.53	≈ 99.06
	Yes	Yes	3	44–46	71–73	≈ 363 105	≈ 988 508	≈ 4 583 534	97.53	≈ 98.75

The '≈' sign indicates mean values over 10 repeated runs in cases, when variation was observed. Results with the smallest number of breakpoints or scaffolds are shown in bold.

split the contig graph into graphs of manageable size by increasing the erosion parameter e.

4.3 Using additional information

To demonstrate the ability of GRASS to utilize various scaffolding information sources, we used two related genomes (see Fig. 5) to help scaffold the *E.coli* assembly: DH10B and BW2952. These genomes were used individually, together and in combination with paired reads. When combining several information sources, care has to be taken in choosing the weights W_r and the erosion threshold parameter e. In individual genome experiments, $W = 100$ and $e = 80$ were chosen to remove links derived from low-quality alignments. In the experiment using only two related genomes (thus no links derived from paired read data) a higher weight was given to the more closely related strain: $e = 70$ and $W_{\mathrm{DH10B}} = 80$, $W_{\mathrm{BW2952}} = 100$ were used for the DH10B and BW2952 strains correspondingly. For experiments combining a single genome with paired reads, $W = 10$ and $e = 4$ were chosen. Finally, $W_{\mathrm{DH10B}} = W_{\mathrm{BW2952}} = 3$ and $e = 5$ were used in the experiment combining all data (including the paired read constraints) to emphasize use of links supported by at least two information sources. When used in the experiment, paired read link weights were set to 1. An SD of 3000 bp was used for links derived from related genomes.

Interestingly, using just related genomes GRASS constructs a smaller number of scaffolds than when only paired reads are used. Table 5 shows, however, that this is achieved at the expense of scaffold accuracy: besides having an increased number of breakpoints, scaffolds constructed based on related genomes alone have a high-total assembly length and, as a consequence, a low-scaffold coverage. The higher than anticipated total assembly length is due to differences in contig distances (i.e. physical distances obtained by aligning contigs to a genome sequence) between the MG1655 strain and the related strains. This is also the reason for the large differences observed between breakpoints at $\Delta = 10$ kb and $\Delta = 500$ bp: while relative order and orientation have been preserved for large parts of the genomes of the considered strains, the exact physical distances have not. This situation is partially alleviated when information from the two genomes is combined, because (i) consistent links (derived from the two genomes) get higher weights after link bundling, and (ii) the more closely related strain BW2952 was given a higher weight. In this case, GRASS is able to further reduce the number of scaffolds without introducing new breakpoints.

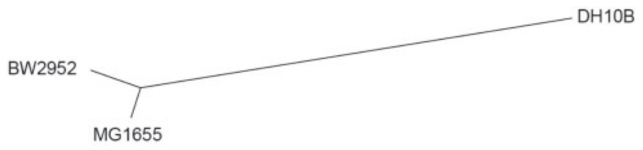

Fig. 5. Phylogenetic tree showing evolutionary distance between the *E.coli* MG1655 strain and two related strains. Genome sequences were obtained from GenBank

Combining paired read data with information from individual related genomes allows for construction of a smaller number of scaffolds with fewer breakpoints than when using these data individually. The results vary between repeated runs of the algorithm, due to inconsistencies between linking information provided by paired reads and related genomes, combined with the stochastic nature of the optimization strategy used for solving the MIQP formulation. Depending on the intermediate solutions found, different contig links are disabled in the optimization process, leading to different final solutions and, thereby to different scaffolds. Table 5, hence, shows a range of scaffold and breakpoint counts, and other results as averages over five repeated runs. This variability is smaller when all data is combined, since a 'voting' approach can be implemented by setting W and e in such a way that all links supported by only a single information source have low weights and are ignored. Using all available information, GRASS reduced the number of scaffolds by 40% compared with just using paired reads, at the expense of introducing a single new breakpoint. The increase in the number of breakpoints is not surprising, as the *de novo* scaffolding information is augmented with links derived for a different (related) organism. The best result on combined data is shown in Figure 4a.

5 CONCLUSION

We presented GRASS, a generic scaffolding algorithm suitable for combining multiple information sources, as well as GRASS+, incorporating a post-processing scaffolding step. Its use was demonstrated by scaffolding genomes based on paired read data and information in related genome sequences, both individually and combined. GRASS achieves the best results when all available scaffolding information is used, as this allows conflicting information from a single source to be ignored when the majority of sources do not support it. Such a mode of operation is supported by the possibility of choosing weights for the individual information sources, combined with the contig link erosion threshold.

We compared GRASS with a number of state-of-the-art scaffolders (SSPACE, MIP and OPERA) on three datasets. GRASS constructs the most accurate scaffolds on all datasets, while keeping the number of scaffolds low. Only SSPACE consistently produces lower numbers of scaffolds, but these are significantly less accurate. The accuracy/contiguity tradeoff displayed by GRASS puts it in a unique niche compared with existing scaffolders.

The current implementation of GRASS supports the use of paired read information and related genomes for scaffolding. However, the algorithm is not limited to any particular set of information sources. We will extend GRASS to allow use of other sources, such as optical restriction maps, RNA-seq and EST data.

ACKNOWLEGEMENT

We would like to thank Bas Dutilh for fruitful discussion and the anonymous reviewers for comments which helped improve the manuscript.

Funding: Plant Green Synthetic Biology Consortium and Kluyver Centre for Genomics of Industrial Fermentation, subsidiaries of the Netherlands Genomics Initiative (NGI).

Conflict of Interest: none declared.

REFERENCES

Auch,A.F. *et al*. (2010) Standard operating procedure for calculating genome-to-genome distances based on high-scoring segment pairs. *Stand. Genomic Sci.*, **2**, 142–148.

Barnett,D.W. *et al*. (2011) BamTools: a C++ API and toolkit for analyzing and managing BAM files. *Bioinformatics*, **27**, 1691–1692.

Beasley,J.E. (1998) Heuristic algorithms for the unconstrained binary quadratic programming problem. *Technical Report*. Management School, Imperial College, London, UK.

Boetzer,M. *et al.* (2011) Scaffolding pre-assembled contigs using SSPACE. *Bioinformatics*, **27**, 578–579.

Dantzig,G.B. (1998) *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, USA.

Dayarian,A. *et al*. (2010) SOPRA: scaffolding algorithm for paired reads via statistical optimization. *BMC Bioinformatics*, **11**, 345.

Delcher,A.L. *et al*. (2002) Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Res.*, **30**, 2478–2483.

Gao,S. *et al*. (2011) Opera: reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. *J. Comput. Biol.*, **18**, 1681–1691.

Henz,S.R. *et al*. (2004) Whole-genome prokaryotic phylogeny. *Bioinformatics*, **21**, 2329–2335.

Hirschberg,D.S. (1975) A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, **18**, 341–343.

Huson,D.H. *et al*. (2002) The greedy path-merging algorithm for contig scaffolding. *J. ACM*, **49**, 603–615.

Huson,D.H. and Bryant,D. (2006) Application of phylogenetic networks in evolutionary studies. *Mol. Biol. Evol.*, **23**, 254–267.

IBM ILOG (2011) ILOG CPLEX: high-performance software for mathematical programming and optimization. Available at http://www.ilog.com/products/cplex. Last accessed date: April 6, 2012.

Kececioglu,J.D. and Myers,E.W. (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.

Langmead,B. *et al*. (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.*, **10**, R25.

Li,H. and Durbin,R. (2009) Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, **25**, 1754–1760.

Li,H. *et al*. (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, **25**, 2078–2079.

Merz,P. and Katayama,K. (2004) Memetic algorithms for the unconstrained binary quadratic programming problem. *BioSystems*, **78**, 99–118.

Miller,J.R. *et al*. (2008) Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, **24**, 2818–2824.

Myers,E.W. *et al*. (2000) A whole-genome assembly of *Drosophila*. *Science*, **287**, 2196–2204.

National Center for Biotechnology Information (2011) Biological Sequence Data Model. In Vakatov,D. (ed.) *The NCBI C++ Toolkit Book (Internet)*. Bethesda, MA, USA.

Nagarajan,N. *et al*. (2008) Scaffolding and validation of bacterial genome assemblies using optical restriction maps. *Bioinformatics*, **10**, 1229–1235.

Nemhauser,G.L. and Wolsey,L.A. (1988) Integer and combinatorial optimization. Wiley-Interscience, New York, NY, USA.

Nesterov,Y. (1997) Quality of semidefinite relaxation for nonconvex quadratic optimization. *CORE Discussion Papers 1997019*. Université Catholique de Louvain, Center for Operations Research (CORE).

Pardalos,P.M. *et al*. (2008) Global equilibrium search applied to the unconstrained binary quadratic optimization problem. *Optim. Meth. Softw.*, **14**, 129–140.

Peng,Y. *et al*. (2010) IDBA – a practical iterative de Bruijn graph *de novo* assembler. *Genome Res.*, **13**, 149–159.

Pop,M. *et al*. (2004) Hierarchical scaffolding with Bambus. *Genome Res.*, **14**, 149–159.

Quinlan,A.R. and Hall,I.M. (2010) BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, **26**, 841–842.

Salmela,L. *et al.* (2011) Fast scaffolding with small independent mixed integer programs. *Bioinformatics*, **27**, 3259–3265.

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

Zerbino,D.R. (2009) Genome assembly and comparison. PhD Thesis, European Bioinformatics Institute, Cambridge, UK.