Bioinformatics, 31(16), 2015, 2632–2638
doi: 10.1093/bioinformatics/btv211
Advance Access Publication Date: 17 April 2015
Original Paper

OXFORD

Sequence analysis

ScaffMatch: scaffolding algorithm based on maximum weight matching

Igor Mandric and Alex Zelikovsky*

Department of Computer Science, Georgia State University, Atlanta, GA 30303, USA

*To whom correspondence should be addressed.

Associate Editor: John Hancock

Received on November 18, 2014; revised on March 4, 2015; accepted on April 10, 2015

Abstract

Motivation: Next-generation high-throughput sequencing has become a state-of-the-art technique in genome assembly. Scaffolding is one of the main stages of the assembly pipeline. During this stage, contigs assembled from the paired-end reads are merged into bigger chains called scaffolds. Because of a high level of statistical noise, chimeric reads, and genome repeats the problem of scaffolding is a challenging task. Current scaffolding software packages widely vary in their quality and are highly dependent on the read data quality and genome complexity. There are no clear winners and multiple opportunities for further improvements of the tools still exist.

Results: This article presents an efficient scaffolding algorithm ScaffMatch that is able to handle reads with both short (<600 bp) and long (>35 000 bp) insert sizes producing high-quality scaffolds. We evaluate our scaffolding tool with the *F* score and other metrics (N50, corrected N50) on eight datasets comparing it with the most available packages. Our experiments show that ScaffMatch is the tool of preference for the most datasets.

Availability and implementation: The source code is available at http://alan.cs.gsu.edu/NGS/?q=content/scaffmatch.

Contact: mandric@cs.gsu.edu

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

1 Introduction

Next-generation sequencing (NGS) is a powerful technology as it can produce millions of short paired-end reads covering whole genome. The cost of producing reads keeps a trend of decreasing making it a very attractive tool for genome sequencing and assembly. Genome assembly from short NGS reads is a challenging problem. Assembled genomes are frequently highly fragmented and consist of contigs of highly variable length. The connectivity information coming from read pairs mapped to contigs can be used to merge them into a *scaffold* which is a set of chains of oriented ordered contigs with estimated gaps between all pairs of adjacent elements. A recent comprehensive evaluation of available software tools shows that the scaffolding problem still does not have an adequate solution (Hunt *et al.*, 2014).

Because of misassemblies in contigs, repeats and chimeric reads, the information about relative ordering and orientation of two contigs connected with a set of read pairs can be contradictory and not reliable. Thus, choosing a wrong subset of read pairs as an evidence of connection between two contigs can result in inferring a wrong relative ordering and/or orientation as well as the gap estimation between them. Edges that comply with the true orientation of contigs and the distance between them are usually called concordant, otherwise discordant edges. The scaffolding problem is usually formulated as an optimization problem on a graph with the vertices corresponding to the contigs and the edges corresponding to the bunches of read pairs connecting contigs. In contrast, the optimization objectives and the methods to find optimal scaffolding with respect to these objectives significantly vary from one tool to another. For example, OPERA (Gao *et al.*, 2011) maximizes the number of concordant edges using dynamic programming, SCARPA (Donmez and Brudno, 2013) minimizes the number of discordant edges using fixed-parameter tractable and bounded algorithm and linear

programming and SILP2 (Lindsay *et al.*, 2014) maximizes number of read pairs agreeing with a chosen contig orientation using non-serial dynamic programming. As all optimization scaffolding formulations are NP-hard (see e.g. Gao *et al.*, 2011), many state-of-the-art scaffolding tools use heuristic approaches exploiting specific properties of the underlying scaffolding graph. SCARPA and SILP2 solve separately the orientation and the ordering problems, MIP (Salmela *et al.*, 2011) and SILP2 use decomposition of the scaffolding graph into biconnected components. SSPACE (Boetzer *et al.*, 2011) uses a simple but powerful greedy heuristic and BESST (Sahlin *et al.*, 2014) uses read pair statistics to filter out spurious edges created by structural errors and starts scaffolding with larger contigs.

In this article, we propose a novel optimization formulation representing scaffolding as a maximum-weight *acyclic* 2-matching (MWA2M) problem. As the Hamiltonian path problem can be reduced to this problem, this formulation is also NP-complete. The presented algorithm ScaffMatch efficiently finds the maximum-weight 2-matching and iteratively destroys all cycles. This approach works very well since, usually, number of cycles is very small.

Our experimental study follows the evaluation of state-of-the-art scaffolders in Hunt *et al.* (2014) performed on five scaffolding datasets [including four from the GAGE project (Salzberg *et al.*, 2012)]. We have run the majority of up-to-date versions of stand-alone scaffolders such as OPERA, SOPRA (Dayarian *et al.*, 2010), SSPACE and MIP as well the recently published ones, SILP2 and BESST. We have also included the results for scaffolding modules of SGA (Simpson and Durbin, 2012) and SOAPdenovo2 (Luo *et al.*, 2012) run independently of de novo assembly following (Hunt *et al.*, 2014). Our matching-based tool ScaffMatch compares favorably with the state-of-the-art tools in terms of widely accepted N50 metric, the metrics introduced in Hunt *et al.* (2014), as well as sensitivity, PPV and *F* score in predicting contig junctions.

2 Methods

Below we describe the problem formulation and algorithmic details in the following main scaffolding steps:

- Preprocessing of read pairs including read mapping, handling repeats and gap estimation for each read pair.
- Scaffolding graph construction with vertices corresponding to contig strands and edges corresponding to read pairs.
- Matching scaffold finding near-maximum weight paths in the scaffolding graph and the corresponding orientation and ordering contigs.
- Insertion of skipped contigs into the matching scaffold.

We conclude with the implementation details of our scaffolding tool.

2.1 Read preprocessing

Each contig has two reverse complement strands, and each read from a pair is mapped to one of the strands. We discard reads aligned to (suspected) repeats. First, we filter out read pairs in which at least one read has multiple alignments. Then for each contig we compute its read coverage and filter out contigs with coverage greater by 2.5σ than the average where σ is the standard deviation of contig coverage. This empirically chosen threshold allows to remove majority of repeats while keeping almost 99% of correct contigs. Although assemblers may give chimeric contigs or produce two contigs for the same genomic region (representing the two haplotypes), ScaffMatch does not attempt to identify or modify any contigs.

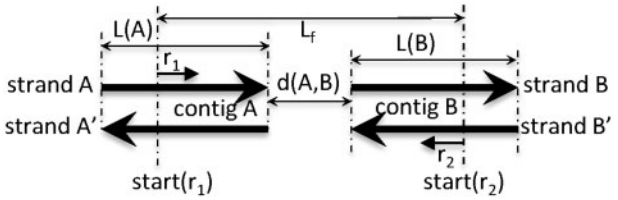

Fig. 1. Gap estimation d is calculated in conformity with the formula: $d = L_f - (L(A) - \text{start}(r_1)) - (L(B) - \text{start}(r_2))$, where L_f is the fragment length, $L(A), L(B)$ are lengths of contigs A and B, start(r_1) (respectively, start(r_2)) is the distance from the starting position of r_1 (respectively r_2) to the beginning of the strand A (respectively, B')

Each read is mapped only to one of the two contig strands (palindromes are discarded). For each read pair connecting two distinct contig strands, we estimate the gap according to Figure 1 [for an alternative gap estimation model see Sahlin *et al.* (2012)]. Among all read pairs connecting the same contig strands, we find a read pair p with the median gap estimation and then bundle p with all read pairs whose gap estimation is at most 3σ away from p's estimation. As we want to keep only reads that agree with each other, the reads outside of this bundle are discarded.

2.2 Scaffolding graph

Each vertex of the scaffolding graph $G = (V, E)$ corresponds to one of the contig strands and each *inter-contig* edge corresponds to a bundle of read pairs connecting two strands of different contigs (Fig. 2). The weight of an inter-contig edge is equal to the size of the corresponding bundle. Also for each contig, we have a *dummy* edge connecting its two strands.

2.3 Matching scaffolding

Ideally, we expect that the scaffolding graph would consist of a set of paths each corresponding to a different chromosome (Fig. 3a). Unfortunately, repeats introduce noisy edges connecting unrelated contigs even from different chromosomes. Additionally, the paths corresponding to chromosomes may skip short contigs (especially contigs which are shorter than the insert length). Therefore, any set of paths passing through all dummy edges in the scaffolding graph G corresponds to a plausible scaffold (Fig. 3b). The most likely scaffold would be supported by the largest number of read pairs. Therefore, we can formulate the following

2.3.1 The scaffolding problem

Given a scaffolding graph G, find a set of paths passing through all dummy edges with maximum total weight of all inter-contig edges.

By setting the weight of each dummy edge to a large number (e.g. number of all read pairs), we reduce the scaffolding problem to the following.

2.3.2 MWA2M problem

Given a weighted graph $G = (V, E, w)$, find a maximum weight acyclic subset of edges $M \subseteq E$, such that each vertex $v \in V$ is incident to at most two edges in M.

The MW2AM of an n-vertex graph G with all edge weights 1 has the weight $n-1$ if and only if G has a Hamiltonian path. Therefore, the MWA2M problem is NP-complete since it includes the Hamiltonian path problem. A similar well-known problem, the maximum weight 2-matching (MW2M), allows chosen edges to form cycles. In contrast to the MWA2M problem, the MW2M problem can be efficiently solved (Pulleyblank, 1980).

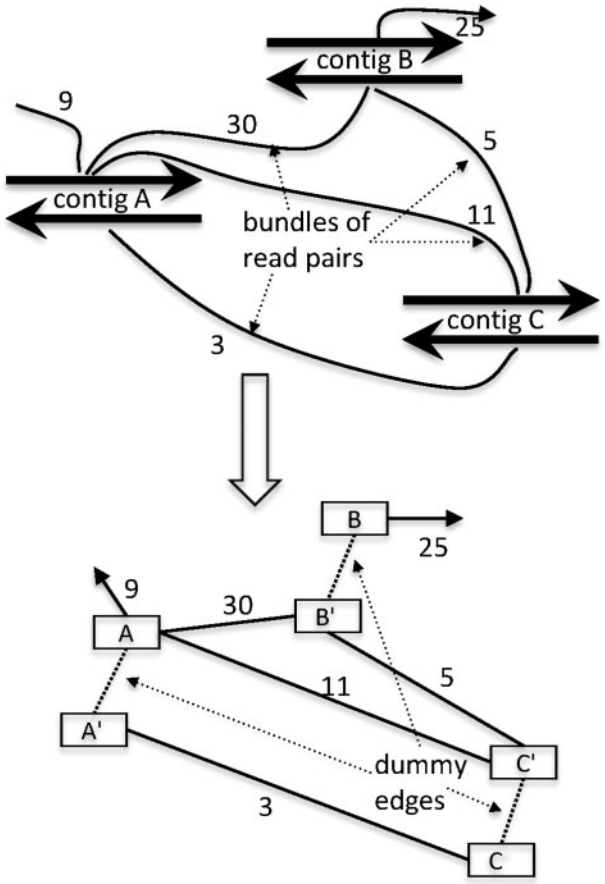

Fig. 2. Contigs *A, B* and *C* with connecting bundles of read pairs and the corresponding scaffolding graph. Each contig is split into two nodes connected with a dummy edge. Each bundle of read pairs corresponds to an inter-contig edge connecting respective strands with the weight equal to the size of the bundle. A plausible scaffold corresponds to the path $C' - C - A' - A - B' - B$ supported by two inter-contig edges CA' and AB'

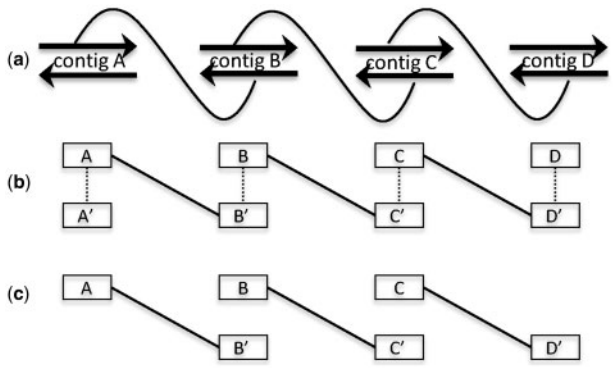

Fig. 3. (**a**) A scaffold $A - B - C - D$: the connection of each pair of adjacent contigs is supported by bundles of read pairs. (**b**) A path $A' - A - B' - B - C' - C - D' - D$ in the scaffolding graph corresponding to the scaffold $A - B - C - D$. (**c**) The matching of the scaffolding graph corresponding to the bunches of read pairs supporting adjacent contigs

2.3.3 Maximum-weight matching heuristic for the MWA2M problem

We propose to almost optimally solve the MWA2M problem with the following iterative heuristic based on the well-known blossom algorithm (Edmonds, 1965) for finding the maximum-weight matching in weighted graphs. It starts with finding the maximum-weight matching M among the inter-contig edges. All the dummy edges also form a matching D. If the union of these two matchings $M \cup D$ does not contain cycles, then the heuristic reaches the optimal collection of paths. Otherwise, a negative weight –1 is assigned to the least weight inter-contig edge in each cycle. The above steps of finding the intercontig matching M and destroying cycles in $M \cup D$ are repeated until the union $M \cup D$ becomes a collection of paths. The output of this heuristic will be called the Matching Scaffold.

In general, the deletion of least-weight edges may significantly reduce (as much as twice in the worst case) the total weight of the collection of paths. Fortunately, the erroneous heavy inter-contig edges are very rare in real data. Our experiments show that for each scaffolding example, there is no more than a single cycle in the initial union $M \cup D$ of the maximum-weight matching M and the dummy edges solution and after the second iteration $M \cup D$ does not contain any cycles at all.

2.3.4 Greedy heuristic for the MWA2M problem

The maximum weight matching can be computed efficiently even for larger genomes. Still the runtime can be dramatically decreased using the Greedy Heuristic repeatedly choosing the heaviest feasible edge, i.e. an edge which does not make a vertex degree higher than 2 and does not form cycles with the previously chosen edges. Note that the Greedy Heuristic picks the *globally* heaviest edge in contrast to greedy scaffolders (such as SSPACE) greedily extending existing chains. We provide an option that allows ScaffMatch to run with the Greedy Heuristic reducing the runtime complexity from $O(n^3)$ to $O(n \cdot \log n)$ as we use max heap in our implementation. Our experiments show that the Greedy Matching performs very well in practice but sacrificing not much in quality to the maximum-weight matching heuristic.

2.3.5 Contig ordering and orientation

The Matching Scaffold is represented by a collection of disjoint chains of contig strands. The sequence of edges along each chain alternates: two strands of the same contig are connected with a dummy edge and two strands of different contigs are connected with an inter-contig edge. When traversing the strands along the paths in the matching scaffold, the order of traversing ends of dummy edges gives us the orientation of the corresponding contigs and the order of traversing inter-contig edges gives us the relative order of contigs.

2.4 Insertion of skipped contigs

The matching scaffold can skip short contigs whose length is less than the read pair insert size. For example, let the true scaffold contain a triple of consecutive contigs A, B and C such that $l_A > l_{\text{ins}},\ l_C > l_{\text{ins}}$, but $l_B \ll l_{\text{ins}}$. Then instead of picking both edges AB and BC, the Matching Scaffold may choose one single edge AC since the edge weight between short contigs depends almost linearly on the length of the contigs. Thus, even though the contig B must follow A in the final scaffold, the weight of the edge between A and B is much smaller than the weight of the edge between A and C, which 'jumps' over B.

Below we describe the insertion of skipped contigs into the matching scaffold (Algorithm 1). A contig is identified as *skipped* only if it is isolated or is a part of a 2-chain in the matching scaffold. Scaffolds with more than two contigs are kept intact. For each skipped contig, we identify the most bundle-supported slot in the matching scaffold satisfying the gap estimations and insert it in this slot (Fig. 4). If several skipped contigs are assigned to the same slot, their relative order and orientation is decided based on the gap

Algorithm 1 Insertion of skipped contigs

$\mathcal{SLOTS} \leftarrow \{\}$
$SKIPPED \leftarrow$ the set of skipped contigs
$M = \{m_1, m_2, ..., m_n\} \leftarrow$ the matching scaffold
$G = (V, E, w) \leftarrow$ the scaffolding graph
$l \leftarrow$ insert length
for all $\mathcal{X} = (X', X) \in SKIPPED$ **do**
  for all $m \in M$ **do**
    if $\exists$ contigs $\mathcal{A} = (A, A'), \mathcal{B} = (B, B') \in m$
    s.t. $(A, X') \in E$ and $(B', X) \in E$ &
      $\text{gap}(\mathcal{A}, \mathcal{X}) + l(\mathcal{X}) + \text{gap}(\mathcal{X}, \mathcal{B}) \leq l$ **then**
        for each edge $e = (\mathcal{C}_i, \mathcal{C}_{i+1}) \in m(\mathcal{A}, \mathcal{B})$ **do**
          if $\text{gap}(\mathcal{A}, \mathcal{C}_i) \leq \text{gap}(\mathcal{A}, \mathcal{X})$ **and**
            $\text{gap}(\mathcal{C}_{i+1}, \mathcal{B}) \leq \text{gap}(\mathcal{X}, \mathcal{B})$ **then**
            $\mathcal{SLOTS}[e, \mathcal{X}]$ += $w(A, X') + w(X, B')$
          end if
        end for
    end if
  end for
end for
for all $\mathcal{X} \in SKIPPED$ **do**
  $e \leftarrow \max\{\mathcal{SLOTS}[e, \mathcal{X}] | e \in E\}$
  insert $\mathcal{X}$ **into** e
end for

estimations as follows. For each skipped contig (X', X), we estimate the distance to the left contig and orient it according to the adjacent strands. Then we sort all contigs with respect to this distance and insert them according to this order.

2.5 Software implementation

Our algorithm is implemented as a stand-alone software tool called ScaffMatch. We separately provide a UNIX shell script for mapping reads to contigs. As a short read aligner, bowtie2 is used (Langmead and Salzberg, 2012). The scaffolder takes as input a fasta file containing the contigs and two SAM files produced by mapping the two read files to the contigs. We keep a small set of mandatory parameters: the mean insert size, the standard deviation and the orientation (forward–reverse, reverse–forward or SOLiD-style) of the paired-end reads. The program outputs a fasta file with scaffolds. We use Networkx python library for graph computations (Hagberg *et al.*, 2005).

3 Results

3.1 Datasets

We validate and compare the scaffolding tools on the collection of scaffolding datasets used in Hunt *et al.* (2014) including four datasets from the GAGE project (Salzberg *et al.*, 2012) [*Staphylococcus aureus*, *Rhodobacter sphaeroides* and *Homo sapiens* (*chr14*)] and one additional dataset *Plasmodium falciparum* following Hunt *et al.* (2014). All contigs were assembled by Velvet (Zerbino and Birney, 2008). The Table 1 gives the parameters of all used scaffolding datasets.

3.2 Quality metrics

The main metric that is used for evaluation of scaffolding tools is N50 (Vezzi *et al.*, 2012). However, this metric may not be representative enough as mentioned in Hunt *et al.* (2014) where a comprehensive evaluation of scaffolders was performed. There state-of-the-art tools were compared based on multiple criteria, such as the number of correct junctions between two adjacent contigs, the number of junctions with incorrect relative order, relative orientation, gap estimation and their combinations (e.g. incorrect relative order + incorrect gap estimation). The scores assigned to the scaffolders, however, can be misleading. For example, MIP on *S.aureus* (using bowtie2) got a high score despite the fact that it joined no contigs. Thus, we introduce an *F*-score-based metric to compare the results of our tool ScaffMatch with other de-novo stand-alone scaffolding tools.

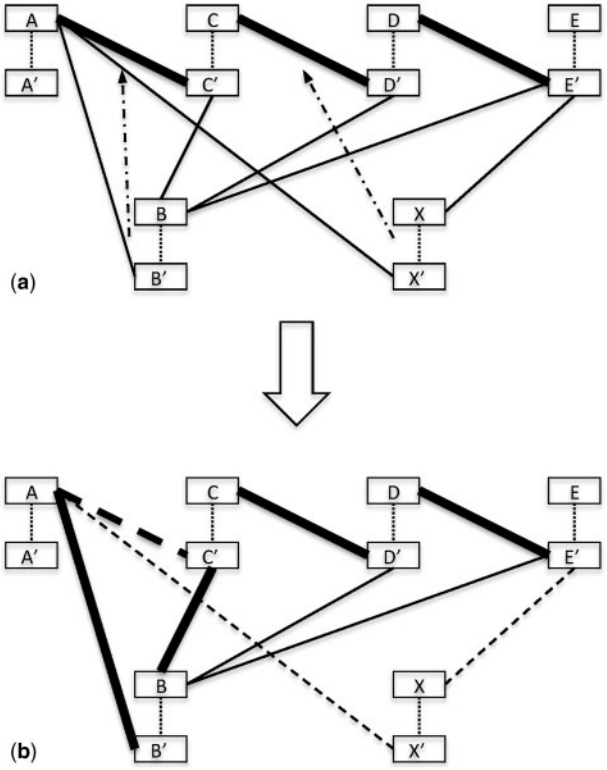

Fig. 4. Insertion procedure. (**a**) The matching scaffold $A - C - D - E$ is obtained with the maximum weight matching; the contig B is connected with edges to all four contigs of the matching, the contig X is connected to A and C; B should be placed between A and C according to the consensus of connecting edges and X should be placed between C and D. (**b**) Since there is a sufficient distance between contigs A and C, B is placed there, i.e. the edge (A, C') from the matching is replaced with (A, B') and (B, C') [the sum of weights of (A, B') and (B, C') is less than the weight of (A, C')]; since there is no sufficient room for X between contigs C and D, the edges (A, X') and (X, E') are removed. The resulted scaffold is $A - B - C - D$

Table 1. Scaffolding datasets

Datasets	Insert size	read. length	No. contigs	No. reads
S.aureus	3600	37	170	3 494 070
R.sphaeroides	3700	101	577	2 050 868
H.sapiens (chr14)				
Short insert size	2865	101	19 936	22 669 408
Long insert size	35 000	80	19 936	2 405 064
P.falciparum				
Short insert size	645	76	9318	52 542 302
Long insert size	2705	75	9318	12 010 344

Table 2. Performance of different algorithms on the scaffolding datasets from GAGE

Dataset	Scaffolder	Correct links	Error links	Skipped contigs	N50	Corr. N50	Sensitivity	PPV	*F* score
S.aureus	ScaffMatch	139	14	23	1 476 925	351 546	**0.832**	0.908	**0.869**
	SSPACE	105	13	13	332 784	261 710	0.629	0.890	0.737
	OPERA	112	11	22	1 084 108	**686 577**	0.671	0.911	0.845
	SOPRA	40	2	7	112 278	112 083	0.240	0.952	0.383
	MIP	0	0	0	46 221	46 221	0	0	0
	SCARPA	77	16	10	112 264	112 083	0.461	0.828	0.592
	SILP2	121	3	34	645 780	284 980	0.725	0.976	0.832
	BESST	112	11	21	**1 716 351**	335 064	0.671	0.911	0.772
	SGA	83	1	10	309 286	309 153	0.497	**0.988**	0.661
	SOAPdenovo2	131	12	13	643 384	621 109	0.784	0.916	0.845
R.sphaeroides	ScaffMatch	482	18	40	**2 547 706**	**2 528 248**	**0.845**	0.964	**0.901**
	SSPACE	357	7	49	109 776	108 410	0.626	0.981	0.764
	OPERA	316	1	23	108 172	108 172	0.554	**0.997**	0.713
	SOPRA	242	15	24	32 232	30 492	0.425	0.942	0.585
	MIP	419	37	16	488 095	487 941	0.735	0.919	0.817
	SCARPA	209	5	23	37 667	37 581	0.367	0.977	0.533
	SILP2	425	24	87	471 077	422 445	0.746	0.947	0.834
	BESST	367	2	15	1 021 151	1 020 921	0.644	0.995	0.782
	SGA	232	1	26	42 825	42 722	0.407	0.996	0.578
	SOAPdenovo2	468	8	26	2 522 483	2 522 482	0.821	0.983	0.895
H.sapiens (chr 14) short insert size	ScaffMatch	12 411	252	3480	131 135	80 329	0.622	0.980	0.761
	SSPACE	9566	43	2754	78 552	77 361	0.487	0.986	0.652
	OPERA	12 291	112	2991	214 972	207 047	0.616	0.991	0.760
	SOPRA	14 761	381	1441	100 768	96 436	0.740	0.975	0.841
	MIP	13 899	954	2735	244 064	**235 731**	0.697	0.936	0.799
	SCARPA	9938	162	1829	58 330	55 760	0.498	0.984	0.661
	SILP2	10 548	124	4918	126 689	77 421	0.529	0.988	0.689
	BESST	7970	355	2165	146 749	80 218	0.400	0.957	0.564
	SGA	9761	6	3214	134 574	133 192	0.490	**0.999**	0.657
	SOAPdenovo2	15 740	390	2378	**282 437**	234 561	**0.790**	0.976	**0.873**
H.sapiens (chr 14) long insert size	ScaffMatch	5938	443	5198	148 412	42 523	**0.298**	0.933	**0.452**
	SSPACE	2750	23	2539	77 832	30 449	0.138	**0.992**	0.242
	OPERA	3687	677	3226	73 477	20 677	0.185	0.845	0.303
	SOPRA	2938	166	2622	79 517	34 750	0.147	0.947	0.255
	MIP	5898	1092	4861	**272 440**	49 800	0.296	0.844	0.438
	SCARPA	1603	31	1466	43 969	17 786	0.080	0.981	0.149
	SILP2	3899	65	3732	74 094	38 810	0.196	0.984	0.326
	BESST	123	13	98	13 815	8828	0.006	0.904	0.012
	SGA	0	0	0	12 211	12 211	0	0	0
	SOAPdenovo2	4516	294	3301	220 644	**86 679**	0.227	0.939	0.365
H.sapiens (chr 14) combined library short + long insert size	ScaffMatch	12 658	348	3874	802 755	195 239	0.635	0.973	0.769
	SSPACE	9249	36	2677	66 271	65 222	0.464	0.996	0.633
	OPERA	12 853	58	3409	**1 692 782**	**1 062 031**	0.645	0.996	0.783
	SOPRA	10 418	238	3322	112 239	75 046	0.523	0.978	0.681
	MIP	8534	696	3213	44 372	31 148	0.428	0.925	0.585
	SCARPA	10 712	161	2376	134 364	106 654	0.537	0.985	0.695
	BESST	8287	286	2347	295 976	114 434	0.416	0.967	0.581
	SGA	9764	3	3214	134 574	133 192	0.490	**0.999**	0.657
	SOAPdenovo2	15 748	382	2575	561 198	447 849	**0.790**	0.976	**0.873**

Various quality metrics have been proposed up to date. Rather than coming up with our own metrics, we have decided to follow the most recent evaluation paper (Hunt *et al.*, 2014), which besides N50 and corrected N50 also reports the number of correctly and erroneously predicted joins between contigs in the reference genome. Following Hunt *et al.* (2014), we do not distinguish between links connecting long and short contigs and contigs from different chromosomes. Let P be the number of potential contigs that can be joined in scaffold which is the number of contigs minus the number of chromosomes, let TP be the number of correct contig joins in the output of the scaffolder (true positives) and FP be the number of erroneous joins (false positives). We compute the following quality metrics: $\text{TPR} = \frac{\text{TP}}{P}$, $\text{PPV} = \frac{\text{TP}}{\text{TP+FP}}$, F score $= \frac{2 \cdot \text{TPR} \cdot \text{PPV}}{\text{TPR+PPV}}$, where TPR is sensitivity and PPV is positive predictive value.

3.3 Evaluation and comparison

ScaffMatch is compared with well-established scaffolders SSPACE, OPERA, SOPRA, MIP, SCARPA, two recently published scaffolders SILP2 and BESST (Sahlin *et al.*, 2014) scaffolding modules of SGA (Simpson and Durbin, 2012) and SOAPdenovo2 (Luo *et al.*, 2012).

Table 3. Performance of different algorithms on the scaffolding datasets for *P.falciparum*. The best values for each dataset are in bold font.

Dataset	Scaffolder	Correct links	Error links	Skipped contigs	N50	Corr. N50	Sensitivity	PPV	*F* score
P.falciparum	ScaffMatch	5648	287	37	**8626**	5872	0.607	0.952	0.741
short insert size	SSPACE	5746	127	12	6011	5845	**0.612**	0.978	**0.757**
	OPERA	3706	116	371	5035	4824	0.398	0.967	0.565
	SOPRA	4897	174	34	4954	4632	0.526	0.966	0.681
	MIP	5544	359	15	6158	5485	0.596	0.939	0.730
	SCARPA	4830	221	38	4912	4628	0.519	0.956	0.673
	SILP2	5496	498	48	3109	2601	0.591	0.917	0.719
	BESST	2632	462	84	7471	3931	0.283	0.851	0.425
	SGA	4940	46	100	5324	5104	0.531	**0.991**	0.691
	SOAPdenovo2	5540	84	47	6234	**5981**	0.596	0.985	0.742
P.falciparum	ScaffMatch	6970	260	1751	41 564	25 380	0.749	0.964	0.843
long insert size	SSPACE	4610	21	1235	17 796	15 553	0.496	0.995	0.662
	OPERA	6257	97	1339	44 667	40 170	0.673	0.985	0.799
	SOPRA	7247	181	656	49 671	44 158	0.779	0.976	**0.866**
	MIP	7754	707	731	88 297	78 672	**0.834**	0.916	0.873
	SCARPA	4882	117	714	14 037	9708	0.525	0.977	0.683
	SILP2	5996	266	2839	45 407	29 399	0.645	0.957	0.771
	BESST	1307	46	327	4133	2813	0.141	0.966	0.245
	SGA	2902	2	652	4438	4096	0.312	**0.999**	0.476
	SOAPdenovo2	7659	351	803	**167 570**	**83 851**	0.635	0.869	0.734
P.falciparum	ScaffMatch	8223	425	654	**78 627**	**47 662**	**0.884**	0.951	**0.916**
combined library	SSPACE	5889	123	76	6383	5982	0.633	0.980	0.769
short + long	OPERA	6434	177	1171	42 450	38 409	0.692	0.973	0.809
insert size	SOPRA	7018	60	171	16 366	15 511	0.754	**0.992**	0.857
	MIP	8082	513	75	56 672	38 704	0.869	0.940	0.903
	SCARPA	7336	370	251	36 945	23 951	0.789	0.952	0.863
	BESST	3929	541	384	25 300	7621	0.422	0.879	0.571
	SGA	4910	44	419	6606	6134	0.528	0.991	0.689
	SOAPdenovo2	5977	228	254	12 076	10 629	0.643	0.963	0.771

SSPACE, OPERA and SOPRA used bowtie (Langmead and Salzberg, 2012) mapping, SOAPdenovo2 used its own mapping and all other scaffolders used bowtie2 mapping. All software has been run with the same versions and options as in Hunt *et al.* (2014) except SILP2 and BESST for which default parameters were used from the respective websites. Results for SILP2 are not given for the combined insert size datasets as it does not have an option to process datasets with multiple insert size.

For computing the number of correct and erroneous links, we used scripts provided in Hunt *et al.* (2014). Note that MIP and SGA did not give meaningful results, respectively, for the *S.aureus* and *H.sapiens* (long insert size).

We compared three different versions of ScaffMatch: ScaffMatch (maximum weight matching with insertion), ScaffMatch_G (greedy matching with insertion) and ScaffMatch_B (maximum weight matching) in the Supplementary Table S1 (see Supplementary Data). ScaffMatch usually has the best *F* score among all three versions. ScaffMatch_G also can be very different from ScaffMatch since it may choose to match completely different contigs. ScaffMatch_B has usually the highest PPV and corrected N50 among all three versions implying that insertion of skipped contigs might be erroneous. Unexpectedly, the number of contigs skipped by ScaffMatch_B is not much greater than for ScaffMatch showing that the solution for the scaffolding/MWA2M problem does not skip over many contigs.

The results for GAGE scaffolding testcases are in Table 2 and results for *P.falciparum* are in Table 3. The entries in the bold font are the best among all 10 scaffolders with respect to the corresponding quality metric. ScaffMatch has the top *F* score for four testcases and the top corrected N50 for two testcases. Additionally, ScaffMatch_B has the top corrected N50 for *S.aureus*. SOAPdenovo2 has the top *F* score for two testcases and the top corrected N50 for three testcases. MIP is a top performer once for *F* score and once for corrected N50. Finally, OPERA is the best in corrected N50 for two testcases (still losing to ScaffMatch_B on *S.aureus*) and SSPACE has the best *F* score for one testcase.

The runtime of all compared scaffolders are in Supplementary Table S2 in Supplementary Data. The runtime growth rate with respect to the dataset size is similar for all scaffolders. The fastest scaffolder is SSPACE, and the slowest is SOPRA.

4 Conclusions

In this article, we present a novel stand-alone scaffolding algorithm ScaffMatch, which is based on representation of the scaffolding problem as the maximum weight acyclic 2-matching. Our experiments show that, unexpectedly, the number of skipped contigs is not significant—for many datasets, the results for ScaffMatch_B are only slightly improved by insertion of skipped contigs in ScaffMatch. The ScaffMatch software implementation is shown to be consistently one of the top performers on majority of the scaffolding datasets.

Funding

This work is supported in part by the NSF Grant IIS-0916401.

Conflict of Interest: none declared.

References

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

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

Donmez,N. and Brudno,M. (2013) Scarpa: scaffolding reads with practical algorithms. *Bioinformatics,* **29**, 428–434.

Edmonds,J. (1965) Paths, trees, and flowers. *Can. J. Math.,* **17**, 449–467.

Gao,S. *et al.* (2011) Opera: reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. *Journal of Computational Biology,* **18**, 1681–1691.

Hagberg,A. *et al.* (2005) Networkx: Python software for the analysis of networks. Technical report, Mathematical Modeling and Analysis. Los Alamos National Laboratory, 2005. http://networkx.lanl.gov (15 April 2015, date last accessed).

Hunt,M. *et al.* (2014) A comprehensive evaluation of assembly scaffolding tools. *Genome Biol.,* **15**, R42.

Langmead,B. and Salzberg,S.L. (2012) Fast gapped-read alignment with Bowtie 2. *Nat. Methods,* **9**, 357–359.

Lindsay,J. *et al.* (2014) Ilp-based maximum likelihood genome scaffolding. *BMC Bioinformatics,* **15**(Suppl 9):S9.

Luo,R. *et al.* (2012) Soapdenovo2: an empirically improved memory-efficient short-read de novo assembler. *Gigascience,* **1**, 18.

Pulleyblank,W.R. (1980) Dual integrity in b-matching problems. *Mathematical Programming Study,* **12**, 176–186.

Sahlin,K. *et al.* (2012) Improved gap size estimation for scaffolding algorithms. *Bioinformatics,* **28**, 2215–2222.

Sahlin,K. *et al.* (2014) Besst-efficient scaffolding of large fragmented assemblies. *BMC Bioinformatics,* **15**, 281.

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

Salzberg,S.L. *et al.* (2012) Gage: a critical evaluation of genome assemblies and assembly algorithms. *Genome Res.,* **22**, 557–567.

Simpson,J.T. and Durbin,R. (2012) Efficient de novo assembly of large genomes using compressed data structures. *Genome Res.,* **22**, 549–556.

Vezzi,F. *et al.* (2012) Feature-by-feature—evaluating de novo sequence assembly. *PLoS One,* **7**, e31002.

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