Exact approaches for scaffolding

Mathias Weller, Annie Chateau, Rodolphe Giroudeau

Weller et al. BMC Bioinformatics 2015, 16(Suppl 14):S2
http://www.biomedcentral.com/1471-2105/16/S14/S2

BMC Bioinformatics

RESEARCH Open Access

Exact approaches for scaffolding

Mathias Weller[1,2*], Annie Chateau[1,2], Rodolphe Giroudeau[1]

From 13th Annual Research in Computational Molecular Biology (RECOMB) Satellite Workshop on Comparative Genomics
Frankfurt, Germany. 4-7 October 2015

Abstract

This paper presents new structural and algorithmic results around the scaffolding problem, which occurs prominently in next generation sequencing. The problem can be formalized as an optimization problem on a special graph, the "scaffold graph". We prove that the problem is polynomial if this graph is a tree by providing a dynamic programming algorithm for this case. This algorithm serves as a basis to deduce an exact algorithm for general graphs using a tree decomposition of the input. We explore other structural parameters, proving a linear-size problem kernel with respect to the size of a feedback-edge set on a restricted version of Scaffolding. Finally, we examine some parameters of scaffold graphs, which are based on real-world genomes, revealing that the feedback edge set is significantly smaller than the input size.

Introduction

During the last decade, a huge amount of new genomes have been sequenced, leading to an abundance of available DNA resources. Nevertheless, most of recent genome projects stay unfinished, in the sense that databases contain much more incompletely assembled genomes than whole stable reference genomes [1]. One reason for this phenomenon is that, for most of the analyses performed on DNA, an incomplete assembly is sufficient. Sometimes it is even possible to perform them directly from the sequenced data. Another reason is that producing a complete genome, or an as-complete-as-possible-genome is a difficult task. Traditionally, producing a complete genome consists of three steps, each of them computationally or financially difficult: the assembly, the scaffolding, and the finishing. The step of scaffolding, on which we focus here, consists of orienting and ordering at the same time the contigs produced by assembly. Many methods have been proposed and the recent activity on the subject shows that it is an active field (see, not exhaustively, [2-8] and Section). A good survey of recent methods can be found in [9] for instance. Since the problem has been proved $\mathcal{NP}$-complete from its first formulation [10], nearly all of these methods propose heuristic approaches. One of them claims that an exact method provides better results [5], however, the authors prepend a heuristic graph simplification before running their exact algorithm.

The approach presented here relies on a combinatorial problem on a dedicated graph, called *scaffold graph*, representing the link between already assembled contigs. The main idea is to represent each contig by two vertices linked by an edge (these "contig-edges" form a perfect matching on the scaffold graph). Other edges are constructed and weighted using complementary information on the contigs. The weight of a non-contig edge uv, with uu', vv' being contig-edges, corresponds to a confidence measure that the uu' contig is succeeded by the vv' contig (oriented as $u' - u - v - v'$). The scaffold graph is a flexible tool to study the scaffolding issues. Indeed, the graph is a syntactical data-structure which may represent several semantics. For instance, the scaffold graphs used for our previous experiments have been built using Next-Generation Sequencing data, namely paired-end reads. However, we also could provide other type of information to compute the weight on the edges, like for instance ancestral support in a phylogenetic context, or comparison to other extent genomes which could be used as multiple references. The way to define the weight on the edges does not change the main goal of our method, which is to

* Correspondence: weller@lirmm.fr
[1]Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM) - Université de Montpellier - UMR 5506 CNRS, 161 rue Ada, 34090 Montpellier, France
Full list of author information is available at the end of the article

determine the optimal ordering and orientation of the contigs, given a specific criterion. It is also possible to mix two or more criteria in order to take several sources of information into account.

We also introduced two structural parameters σ_p and σ_c representing the respective numbers of linear and circular chromosomes sought in the genome. Those parameters seems quite artificial at a first sight, but they are well-motivated as follows. First, in a number of species, genomes are hard to assemble with classical methods because of an heterogeneous chromosomal structure or difficulties to perform classical assembly. This is the case, for instance, with the micro-chromosomes in the chicken genome [11]. Also, when scaffolding metagenomic data, one has to deal with very complex genomic structures [12]. We hope that relaxing the classical model with one chromosome, mixed with the flexibility of the scaffold graph, could help handling such complex situations. The second motivation to introduce those parameters came from the desire to explore an intermediary case between the very classical and studied $\mathcal{NP}$-complete [13]TRAVELING SALESMAN PROBLEM (TSP) and a totally free structure, in which case optimizing a covering by paths and cycles is polynomial-time solvable [14]. The SCAFFOLDING problem then consists of finding (at most) σ_p paths and σ_c cycles that cover all contig-edges while maximizing the total weight. Previously, we proved that SCAFFOLDING is $\mathcal{NP}$-complete, even under restricted conditions [15], developed polynomial-time approximation algorithms [15,16] and evaluated them experimentally [17]. In this paper, we continue to explore this problem, as well as the structural properties of the scaffold graph. This exploration aims to develop an efficient method, that could be used both to refine the ratio analysis on previously designed heuristics and to handle cases of small genomes with better quality.

We present both theoretical and practical results, as well as our experimental findings. The paper is organized as follows: we relate a general state-of-the-art and methodological comparison with existing methods and our previous work in Section. In Section, we formally introduce the problem and some notations and definitions. In Section, we show that the general SCAFFOLDING problem is polynomial-time solvable on trees, and present a dynamic programming algorithm. We extended this algorithm in Section 11 to a *parameterized* algorithm with respect to the *treewidth* of the input, that is, this algorithm runs in polynomial time, on scaffold graphs that are sparse, *i.e.* treelike. In Section 11, we lay the foundation to developing effective preprocessing routines for SCAFFOLDING, by presenting a set of reduction rules whose application shrinks input instances of a derived problem, RESTRICTED SCAFFOLDING. We show that the size of the resulting instances can be bounded linearly in their feedback edge set number, thus proving a problem kernel for this parameter. Finally, in Section 11, we present some experimental analysis of scaffold graphs that are derived from real-world genomes.

Related work

Genome scaffolding is an intrinsically complex problem, and was studied through several computation models and tools. The first model, presented in [10], leads to its classification as an $\mathcal{NP}$-complete problem. This work also proposes the first greedy approach. The scaffolding was then performed using genomic fragments coupled by mate-pairs [18], which differ from the paired-end reads essentially by their larger insert-size (The insert-size is the gap, in base pairs, between two fragments constituting a pair of reads), and their lower covering depth, so the size of the data and the organization of the graph may differ from actual Next-Generation Sequencing data using paired-end reads. A greedy-like approach is also used in SSPACE [19], iteratively combining the longest contigs. Conversely, Gao et al. present a dynamic-programming-based exact approach (OPERA) which provides higher-quality scaffolds than existing heuristic approaches [5]. However, their algorithm runs in $O(n^k)$ time (where k is the "library width") which quickly grows out of reasonable proportions. Thus, to apply this method on real data, they prepend a graph contraction procedure to limit the size of the input. In SOPRA [4], Dayarian et al. introduce a removal procedure for problematic contigs, and separate the orientation step from the linking step. The orientation step uses a simulated annealing technique for regions of high complexity. In GRASS [3], a genetic algorithm is provided, using mixed integer programming (MIP) as in [6] and an expectation-maximization process to counter the intractability of the MIP model. They obtain results which are intermediary in quality between SSPACE and Opera. In SCARPA [2], the two problems of orienting and ordering the contigs are separated, as in SOPRA. Transforming the graph such that the contig orientation problem has a feasible solution is a fixed-parameter tractable problem with respect to the number of nodes to remove [20]. The authors first use the corresponding algorithm to pre-process the graph and exhibit an optimal relative orientation of the contigs. Then, pre-oriented contigs are ordered using a heuristic, and the removal of articulation vertices is used to limit the size of the connected components. Misassembled contigs are detected and removed. In BESST [21], the authors refine the previous approaches by considering the library insert size distribution to filter relevant linking information, leading to an almost linear scaffold graph, which is easy to treat. The scalability of these approaches is not always proven, and the general increase of the size and amount of available data brings real efficiency questions. Recently, some methods also use

phylogenetic or comparative genomic approaches via rearrangement analysis to perform scaffolding on ancient or extant genomes when a set of closely related species genomes are available (see for instance [22-24]). This encourages the mix of multiple sources of information for scaffolding a given genome.

The scaffolding problem have been extensively studied in the framework of complexity and approximation. In [15,16], the authors proved that the problem is $\mathcal{NP}$-complete even if the scaffold graph is a planar bipartite graph. They also proposed some lower bounds for exact exponential-time algorithms for SCAFFOLDING according to the EXPONENTIAL-TIME HYPOTHESIS [25]. Finally, they proved that the minimization version of SCAFFOLDING (seeking a *minimum-weight* cover of the scaffold graph) is unlikely to be approximable within any constant ratio, even if the scaffold graph is a completed bipartite graph [15]. On the positive side, two polynomial-time factor-3-approximation algorithms for the maximization version have been designed. The first is an $O(n^2 \log n)$-time greedy algorithm, the second is based on the computation of a maximum matching ($O(n^3)$-time). The theoretical aspects of the scaffold graph have been completed by extensive experimental results [17] for the maximization version on simulated and real datasets.

The main contribution of the present paper lays in both the algorithmic exploration of exact methods for the scaffolding problem with structural parameters, and the initiation of a discussion on the scaffold graph properties. Indeed, the knowledge of those properties may lead to algorithmic improvement, as well as the detection of putative errors in assemblies.

Definitions

Scaffolding problem. The central combinatorial object we are working with in this work is the "scaffold graph" (see also [16,17]).

Definition 1 (Scaffold graph) *A* scaffold graph *is a pair* (G, M) *of a graph* $G = (V, E, w)$ *with an even number of vertices, and a weight function* $w : E \to \mathbb{N}$ *on its edges, and a perfect matching* M *on* G.

Notice that this model is close to what previous work call scaffold graph, except that this graph is not a directed graph, and contigs are represented by edges instead of vertices. Figure 1 shows an example of a scaffold graph.

Graph Theory. Slightly abusing notation, we sometimes consider paths as sets of edges. Furthermore, for a matching M and a vertex u, we define $M(u)$ as the unique vertex v with $uv \in M$ if such a v exists, and $M(u) = \perp$, otherwise. A path p is *alternating* with respect to a matching M if, for all vertices u of p, also $M(u)$ is a vertex of p. If M is clear from context, we do not mention it explicitly. For a graph $G = (V, E)$ and a vertex set $V' \subseteq V$, let $G[V']$ denote the subgraph of G induced by V' and let $G - V' := G[V \setminus V']$. For $S \subseteq E$, let $G - S := (V, E \setminus S)$ and, for any edge or vertex x, we abbreviate $G - \{x\} =: G - x$. For a set of pairs S,

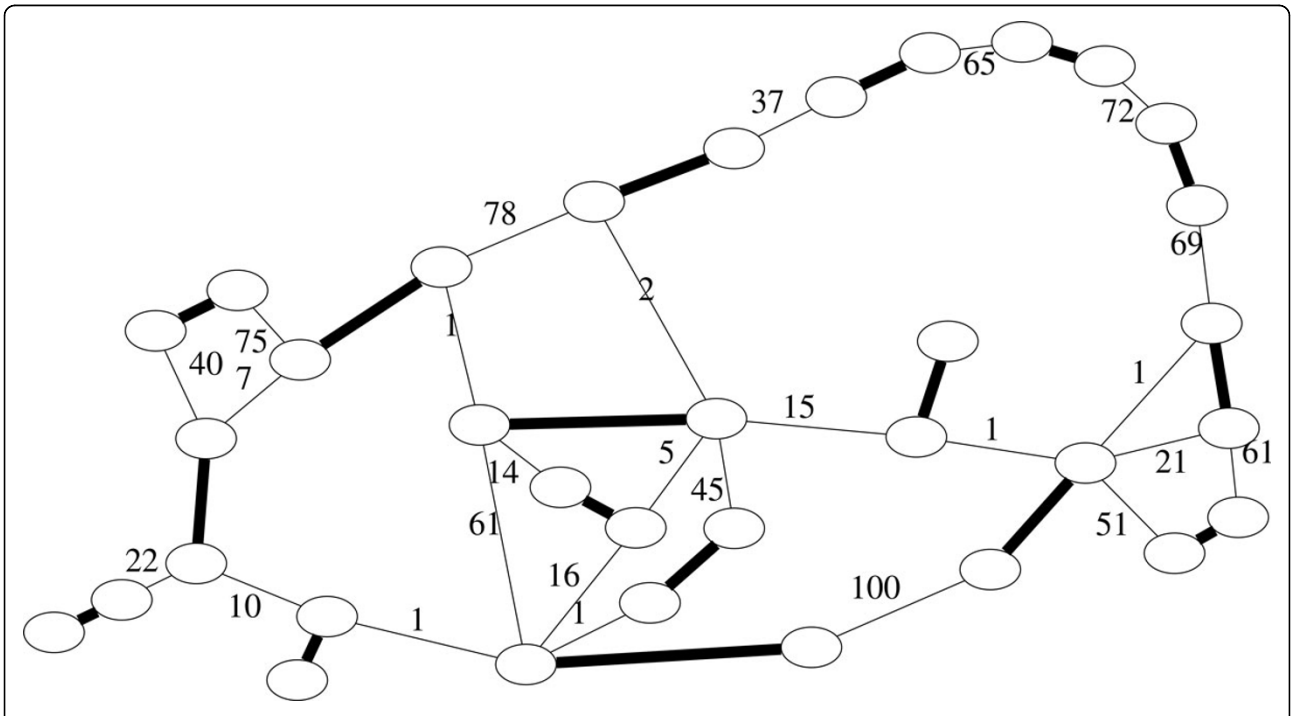

Figure 1 A scaffold graph with 17 contigs (bold edges) and 26 (weighted) links between them, corresponding to the ebola data set (see Section 11).

we let Gr(S) denote the graph having S as edgeset, that is, Gr(S) := ($U_{e\in S}$ e, S). For a function $\omega : E \to \mathbb{N}$ and a set $S \subseteq E$, we abbreviate $\Sigma_{e\in S}\ \omega(e) =: \omega(S)$. Let $G = (V, E)$ be a graph with a matching M and let S be a matching in $G - M$. The number of paths (resp. cycles) in $G[S \cup M]$ is denoted by $||S||_p^{G,M}$ (resp. $||S||_c^{G,M}$). If all paths in $G[S \cup M]$ are alternating with respect to M, then, we call S a $||S||_p$-$||S||_c$-*cover* (or simply *cover*) for G with respect to M (we will omit M if its clear from context). If $||S||_c = 0$, we may also refer to S as a $||S||_p$-*path cover* (or simply *path cover*). The general scaffold problem is expressed as follows (see also [16,17]): SCAFFOLDING (SCA)

Input: A scaffold graph $(G = (V, E, w), M)$, $\sigma_p \in \mathbb{N}$, $\sigma_c \in \mathbb{N}$, $k \in \mathbb{N}$

Question: Is there a σ_p-σ_c-cover S for G with respect to M with $\omega(S) \geq k$?

Tree Decompositions. A tree decomposition of a graph $G = (V, E)$ is a pair $(T = (V^T, E^T), X : V^T \to 2^V)$ such that (1) for all $uv \in E$, there is some $i \in V^T$ with $uv \subseteq X(i)$ and (2) for all $v \in V$, the subtree $T_v := T[\{X(i) \mid v \in X(i)\}]$ is connected. We call the images of X "bags" and the size of the largest bag minus one is the *width* of the decomposition. A decomposition of minimum width for G is called the *optimal* for G and its width is called the *treewidth* of G. It is a folklore theorem that each graph G has an optimal tree decomposition (T, X) that is *nice*, that is, each bag $X(i)$ is one of the following types:

Leaf bag: i is a leaf of T and $X(i) = \emptyset$,

Introduce vertex v **bag:** i is internal with child j and $X(i) = X(j) \cup \{v\}$ with $v \notin X(j)$,

Forget v **bag:** i is internal with child j and $X(i) = X(j) - v$ with $v \in X(i)$,

Introduce edge uv **bag:** i is internal with child j and $uv \in E$ and $uv \subseteq X(i) = X(j)$,

Join bag: i is internal with children j and ℓ and $X(i) = X(j) = X(\ell)$.

Additionally, each edge $e \in E$ is introduced exactly once. Given a width-tw tree decomposition, we can obtain a nice tree decomposition of width tw in $n \cdot$ poly(tw) time [26].

Parameterized Algorithmics. Parameterized complexity theory challenges the traditional view of measuring the running times exclusively in the input size n. Instead, we aim at identifying a parameter of the input that we expect to be small (much smaller than n) in all instances that the application at hand may produce. We then focus on developing algorithms whose exponential part can be bounded in this parameter (see [27] for details). Parameterized complexity allows to prove performance guarantees for preprocessing algorithms: a polynomial-time algorithm that, given an instance x with parameter k, computes an instance x' with parameter $k' \leq k$ such that x is a yes instance if and only if x' is a yes instance, is called *kernelization* and the result x' is the *kernel*. It is common to present a kernelization by showing various *reduction rules* that "cut away" the easy parts of the input and, when applied exhaustively to the input, shrink it enough to prove the size bounds.

Scaffolding on trees

Towards developing an algorithm for SCAFFOLDING that runs fast on graphs that are "close to trees", we consider a strategy to solve SCAFFOLDING in case the input graph is a tree. We use a bottom-up dynamic programming approach that computes for each vertex v starting with the leaves, the best possible solutions for the subtree rooted at v. For ease of presentation, we thus consider the input tree T to be rooted arbitrarily with r denoting the root vertex. Note that the solution cannot contain any cycles in this case.

At each vertex, the dynamic programming algorithm needs to decide how many paths should be covered in each of the subtrees of its children. Seeing that it is infeasible to try all combinations, we employ, again, a dynamic programming strategy solving this problem for a given vertex v: We order the children of each vertex v arbitrarily and let s_v denote the sequence of children of v with $s_v[j]$ denoting the j^{th} child of v and $s_v[1..j] := U_{i\leq j}\ s_v[\,j]$. Then, we update the global dynamic programming table of v in order of s_v. Thus, when we consider the j^{th} child u of v, we only have to split the paths to be covered between the two subtrees T_u and the union of all previously considered subtrees rooted at children of v (that is, $T\,[U_{\ell<j}\ T_{s_v\,[\ell]} \cup \{v\}]$).

In the following, for a vertex v, let $C(v)$ denote its children and par(v) its parent (or $\perp$ if $v = r$), and let T_v denote the subgraph of T that is rooted at v. For $u, v \in V$, we define $T_u + T_v := T\,[V(T_u) \cup V(T_v)]$ and $T_u + v := T\,[V(T_u) \cup \{v\}]$. Let $v \in V$ and $1 \leq j \leq |C(v)|$. Then, we define $T_j^v := \sum_{i\leq j} T_{s_v[i]} + v$. Finally, we abbreviate $s_v := s_v[1..0] := \perp$.

Algorithm 1: Algorithm to fill the dynamic programming table.

1 $I \leftarrow$ leaves of T;
2 while $\exists v \in V \setminus I$ *s.t.* $C(v) \subseteq I$ **do**
3 **foreach** $1 \leq j \leq |C(v)|$ *with* $u := s_v[j]$ **do**
4 **foreach** $i \leq \sigma_p$ **do**
5 **if** $uv \in M$ **then**
6 $[0, i, j]_v \leftarrow \max_{\alpha\in\{0,1\}} \max_{\ell\leq i-(1-\alpha)}[\alpha, \ell, \infty]_u + [0, i - (\ell - \alpha + 1), j - 1]_v$;
7 $[1, i, j]_v \leftarrow \max_{\alpha\in\{0,1\}} \max_{\ell\leq i}[\alpha, \ell, \infty]_u + [1, i - (\ell - \alpha), j - 1]_v$;
8 **else**
9 $[0, i, j]_v \leftarrow \max_{\ell\leq i} \max_{\alpha\in\{0,1\}}[\alpha, \ell, \infty]_u + [0, i - \ell, j - 1]_v$;
10 $[1, i, j]_v \leftarrow \max_{\ell\leq i} \begin{cases} \max_{\alpha\in\{0,1\}}[\alpha, \ell, \infty]_u + [1, i - \ell, j - 1]_v; \\ \omega(uv) + [0, \ell, \infty]_u + \begin{cases} [0, i - (\ell - 1), j - 1]_v & ;\text{if } w \in s_v[1..j] \\ [0, i - \ell, j - 1]_v & \text{otherwise} \end{cases} \end{cases}$
11 if $v = r$ **then return** $\max_{c\in\{0,1\}}[c, \sigma_p, \infty]_r$**else** $I \leftarrow I \cup \{v\}$;

Semantics: *For* $(c, i, j, v) \in \{0, 1\} \times [\sigma_p] \times [|C(v)|] \times V$ *we let* $[c, i, j]_v$ *denote the maximum weight of an* i*-0-cover* S *for* T_j^v *such that* v *is incident with exactly* c *edges of* S*. We abbreviate* $[c, i, \infty]_v := [c, i, |C(v)|]_v$.

We maintain a set I of initialized vertices and, as soon as r is initialized, the algorithm stops. Thus, we assume $r \notin I$. Finally, the maximum weight of a σ_p-0-cover for T can be computed by $\max_{c \in \{0,1\}}[c, \sigma_p, \infty]_r$.

Lemma 1 *Algorithm 1 is correct, that is, for all* $(i, j, c, v) \in [\sigma_p] \times [|C(v)|] \times \{0, 1\} \times V$ *and for any maximum-weight* i*-0-cover* S *for* T_j^v *with respect to* M *such that* v *is incident with exactly* c *edges of* S *we have* $[c, i, j]_v = \omega(S)$.

Concerning the running time, we note that the body of the loop in line 3 is executed exactly once per vertex and, hence, lines 6,7,9, and 10 are executed σ_p times per vertex. As they compute the maximum over σ_p values, the whole algorithm runs in $O(n \cdot \sigma_p^2)$ time.

Corollary 1 SCAFFOLDING*on trees can be solved in* $O(n \cdot \sigma_p^2)$ *time.*

Scaffolding with respect to treewidth

In this section, we develop a parameterized algorithm with respect to the structural parameter "treewidth", solving SCAFFOLDING in $O(\mathrm{tw}^{\mathrm{tw}})$ poly(n, σ_p, σ_c) time. It is based on using a dynamic programming table to keep track of solutions that interact with the bags of a tree decomposition in a certain way (see Figure 2). Since we store for each type of interaction only the best solution and the number of interactions can be bounded in the treewidth, we arrive at the claimed bounds.

To present our algorithm, we use special permutations (involutions) to model matchings that allow reflexive pairing (that is, matching a vertex with itself). Thus, slightly abusing notation, we will consider permutations as sets of pairs. We denote the subset of reflexive pairs of a permutation P by P^1 and the subset of non-reflexive pairs by P^2. Then, $\mathrm{Gr}(P) := \mathrm{Gr}(P^2)$. For permutations P and Q, we define $P \sqcap Q$ as the set of pairs uv such that $P(x) = \perp \oplus Q(x) = \perp$ for all $x \in uv$ and there is a u-v-path in $\mathrm{Gr}(P \cup Q)$. Furthermore, for a function $d : A \to B$ and $(x, y) \in A \times B$, we define $d[x \to y]$ as the result of setting $d(x) := y$ (that is $(d \setminus (\{x\} \times B)) \cup \{(x, y)\}$). Here, $d[x \to \perp]$ means to remove x from the domain of d. Let $T = (\chi, E^T)$ be a tree decomposition of G with root $X(r) \in \chi$. For a bag $X(i)$, let G_i denote the subgraph of G that contains exactly those edges of G that are introduced in a bag of the subtree of T that is rooted at $X(i)$ and let $G_i^S := G_i[S \cup M]$.

A table entry for the bag $X(i)$ will be indexed by (i) a function $d : X(i) \to \{0, 1, 2\}$, (ii) a permutation P with $\bigcup_{uv \in P} uv = d^{-1}(1)$ and(iii) integers $p \le \sigma_p$ and $c \le \sigma_c$. See Figure 2 for an illustration of d and P. An entry will have the following semantics:

Definition 2 *Let* $i \in V^T$*. We call a set* $S_i \subseteq E(G_i) \setminus M$ eligible *with respect to a tuple* (d, P, p, c, i) *if*

1 each vertex $v \in X(i)$ *has degree* $d(v)$ *in* $G_i^{S_i}$,
2 for each $uv \in P$*, if* $u \neq v$*, then there is a* u-v*-path in* $G_i^{S_i}$ *and, if* $u = v$*, then there is a path* q *of non-zero-length in* $G_i^{S_i}$ *that contains* u *and avoids* $d^{-1}(1) - u$ (*we say that* q *is* dangling from u).
3 $G_i^{S_i}$ *contains* p *paths and* c *cycles that do not intersect* $d^{-1}(1)$,

Semantics: *A table entry* $[d, P, p, c]_i$ *is the maximum weight of any set that is eligible with respect to* (d, P, p, c, i).

Then, we can read the maximum weight of a solution S for G from $[\emptyset, \emptyset, \sigma_p, \sigma_c]_r$.

Given a nice tree decomposition and a bag $X(i)$ with children $X(j)$ and $X(\ell)$ (possibly $j = \ell$), we compute $[d, P, p, c]_i$ depending on the type of the bag $X(i)$ (entries that are not mentioned explicitly are set to $-\infty$):

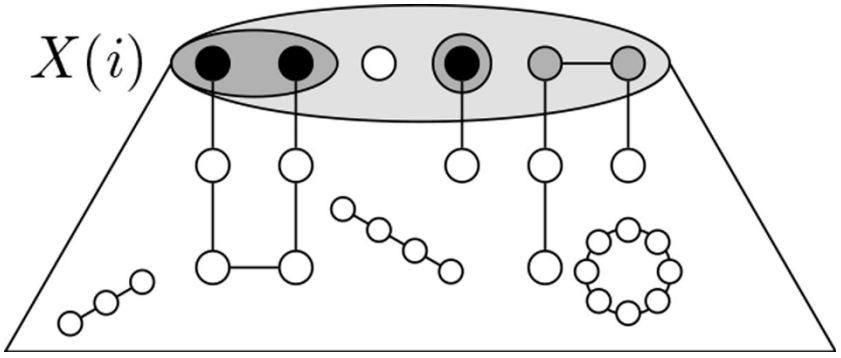

Figure 2 Setup of the dynamic programming on tree decompositions. A vertex $u \in X(i)$ can have degree $d(u) = 0$ (white circle), $d(u) = 1$ (black circle), or $d(u) = 2$ (gray circle) in G_i^S. Vertices with d(u) = 1 are always incident with paths in G_i^S (gray ellipse), forming a pairing (a permutation) P on them.

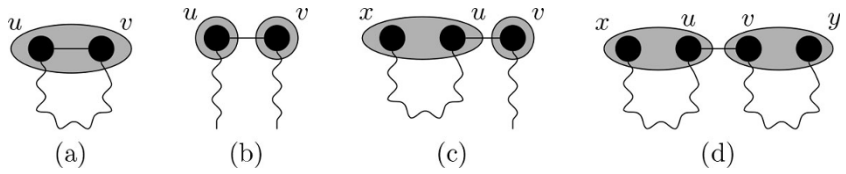

Figure 3 The four cases for introducing an edge *uv* with *d*(*u*) = *d*(*v*) = 2 in the dynamic programming for *X*(*i*): (a) there is a *u*-*v*-path in G_j^S, (b) *u* and *v* have dangling paths in G_j^S, (c) there is a *u*-*x*-path and a dangling path on *v* in G_j^S, or (d) there is a *u*-*x*-path and a *v*-*y*-path in G_j^S.

Leaf bag: Set $[\emptyset, \emptyset, 0, 0]_i := 0$.

Introduce vertex *v*: Newly introduced vertices do not have introduced edges yet. Thus, we force the degree of *v* in G_i^S to 0: Formally, let $[d, P, p, c]_i := [d[v \to \perp], P, p, c]_j$ if $d(v) = 0$ and ∞, otherwise.

Forget vertex *v*: A vertex *v* that we forget in bag *X*(*i*) can have degree 0,1, or 2 in G_j^S. If it has degree 1, then there is a path *q* dangling from it. If *q* ends in some other vertex $u \in X(i) \setminus \{v\} = X(j)$, then, the permutation for *X*(*i*) contains *uu* and the permutation for *X*(*j*) contains *uv*. Otherwise, *q* is dangling from *v* in G_j^S so the permutation for *X*(*j*) contains *vv*. Formally, let

$$[d, P, p, c]_i := \max \begin{cases} \max\limits_{uu \in P} [d[v \to 1], \quad (P - uu) + uv, p, c]_j, \\ [d[v \to 1], P + vv, p - 1, c]_j, \\ \max\limits_{x \in \{0,2\}} [d[v \to x], P, p, c]_j. \end{cases}$$

Introduce edge *uv*: Let $d(u), d(v) \geq 1$ and, by symmetry, let $d(u) \geq d(v) \geq 1$. We define a value *z* (representing the assumption that *uv* is in *S*) as follows. Let $d' := d[u \to d(u) - 1, v \to d(v) - 1]$, that is, we let d' be the result of decreasing both $d(u)$ and $d(v)$ by one.

Case 1 $d(u) = d(v) = 2$. Then, *P* avoids *u* and *v*. Since we assume $uv \in S$, this means that *u* and *v* have dangling paths q_u and q_v in $G_i^S - uv = G_j^S$ that both intersect $d'^{-1}(1)$. If $q_u = q_v$, then adding *uv* to *S* closes a cycle in G_i^S and the permutation for *X*(*j*) contains *uv* (see Figure 3(a)). Otherwise, $q_u \neq q_v$. Then, if q_u intersects $d'^{-1}(1) \setminus \{u, v\}$ in a vertex *x*, then the permutation for *X*(*j*) contains *ux* (see Figure 3(c) and Figure 3(d)), otherwise, it contains *uu* (see Figure 3(b)). Likewise, if q_v intersects $d'^{-1}(1) \setminus \{u, v\}$ in a vertex *y*, then the permutation for *X*(*j*) contains *vy*, otherwise, it contains *vv*. Note that, if both q_u and q_v intersect $d'^{-1}(1) \setminus \{u, v\}$ (see Figure 3(d)), then we have $xy \in P$. Note further that, if neither q_u nor q_v intersects $d'^{-1}(1) \setminus \{u, v\}$ (see Figure 3(b)), then $q_u \cup q_v \cup \{uv\}$ is a path in G_i^S that does not intersect $d^{-1}(1)$ and, thus, we have to decrease the number *p* of such paths we are looking for in G_j^S. Formally, let

$$z := \max \begin{cases} [d', P + uv, p, c - 1]_j, \\ [d', P \cup \{uu, vv\}, p - 1, c]_j, \\ \max\limits_{xx \in P} \begin{cases} [d', (P - xx) \cup \{ux, vv\}, p, c]_j, \\ [d', (P - xx) \cup \{uu, vx\}, p, c]_j, \end{cases} \\ \max\limits_{xy \in P} \{[d', (P - xy) \cup \{ux, vy\}, p, c]_j. \end{cases}$$

Case 2: $d(u) = d(v) = 1$. Then, both *u* and *v* are not incident to any edges in G_j^S and, in $G_j^{s_j}$, there is just the edge *uv* incident to both. Thus, we set *z* only if $uv \in P$. Formally, let $z := [d', P - uv, p, c]_j$ if $uv \in P$.

Case 3: $d(u) = 2$, $d(v) = 1$. Then, there is a path *q* containing *uv* and ending in *v* in G_i^S. If *q* ends in a vertex *x* in $d^{-1}(1) - v$, we have $vx \in P$ and the permutation for *X*(*j*) contains *ux*. Otherwise, we have $vv \in P$ and the permutation for *X*(*j*) contains *uu*. Note that, since $v \in d^{-1}(1)$, we know that $P(v) \neq \perp$. Formally, for $vx \in P$, let $z := [d', (P - vv) + uu, p, c]_j$, if $v = x$ and $z := [d', (P - vx) + ux, p, c]_j$, otherwise. Finally, let $[d, P, p, c]_i := z$ if $uv \in M$ and $[d, P, p, c]_i := \max\{z + \omega(uv), [d, P, p, c]_j\}$, otherwise.

Join: The join bag *X*(*i*) "glues" the (disjoint) partial solutions of its children together at the vertices of $X(i) = X(j) = X(\ell)$. In particular, the degrees in G_j^S and in G_ℓ^S have to add up to *d*. Furthermore, the permutations P_1 and P_2 for *X*(*j*) and *X*(ℓ), respectively, have to "fit" *P*: For example, let $uv \in P_1$ and $vw \in P_2$, implying that there are paths q_j and q_ℓ in G_j^S and G_ℓ^S, respectively, that are connecting *u* and *v* and *v* and *w*, respectively. Then, in G_i^S, there is a single path $q_j \cup q_\ell$ connecting *u* and *w* and containing *v* (with $d(v) = 2$). Finally, the numbers of paths and cycles have to "fit" *p* and *c*: For example, if the permutations for both *X*(*j*) and *X*(ℓ) contain *uu* (that is, $u \in (P_1 \cap P_2)^1$), then G_i^S contains a path containing *u* that is neither in G_j^S nor in G_ℓ^S. On top of this, the remaining paths must be distributed among G_j^S and G_ℓ^S. Formally, let

$$[d, P, p, c]_i := \max_{\substack{d_1, d_2 : X(i) \to \{0,1,2\} \\ \forall v \in X(i), d_1(v) + d_2(v) = d(v)}} \max_{P_1, P_2 \; P = P_1 \sqcup P_2} \max_{\substack{p_1, p_2, c_1, c_2 \\ p_1 + p_2 + |(P_1 \cap P_2)^1| = p \\ c_1 + c_2 + |(P_1 \cap P_2)^2| = c}} \left\{ \begin{matrix} [d_1, P_1, p_1, c_1]_i \\ +[d_2, P_2, p_2, c_2]_\ell \end{matrix} \right\}$$

Lemma 2 *The described algorithm is correct, that is, the computed value* $[d, P, p, c]_i$ *corresponds to the semantics.*

Theorem 1 SCAFFOLDING *can be solved in* $O(\mathrm{tw}^{\mathrm{tw}} \cdot \sigma_p \cdot \sigma_c \cdot n)$ *time, given a width-*tw *tree decomposition of the input instance.*

Kernel for restricted scaffolding

Towards developing effective preprocessing for SCAFFOLDING, we consider a more restricted problem variant, where all paths and cycles of the solution have to be of certain, respective lengths. $\mathcal{NP}$-hardness of this variant can be inferred with the same reduction as used for SCAFFOLDING[15].

RESTRICTED SCAFFOLDING (RSCA)
Input: G with perfect matching M, $\omega : E \to \mathbb{N}$ with $\omega(M) = 0$, $\sigma_p \in \mathbb{N}$, $\sigma_c \in \mathbb{N}$, $\ell_p \in \mathbb{N}^*$, $\ell_c \in \mathbb{N}^*$, $k \in \mathbb{N}$
Question: $\exists_{X \subseteq E \setminus M}$ s.t. $G - X$ is a collection of σ_p alternating paths, each of length ℓ_p,b and σ_c alternating cycles, each of length ℓ_c, and $\omega(E) - \omega(X) \geq k$?

The length l_p denotes the number of edges in the paths. It is necessarily odd in a solution of RESTRICTED SCAFFOLDING. In the following, we show that RESTRICTED SCAFFOLDING admits a linear-size problem kernel with respect to the parameter FES (feedback edge set), which is the size of a smallest set of edges whose deletion leaves an acyclic graph. To this end, we present a number of intuitive polynomial-time executable reduction rules that shrink the input graph. The first two rules shrink treelike structures of the instance while the remaining rules contract long chains.

For a u-v-path p, we call u and v its *outer vertices* while all other vertices of p are *inner vertices*. Let V° be the set of vertices that are in some cycle in G and let $G^\circ = G[V^\circ]$. Let $G^* = G[V^*]$ denote the convex hull of G°, that is, V^* is the set of vertices on some shortest path between some vertices $u, v \in V^\circ$. For each $v \in V^*$, the tree rooted at v that is incident with v in $G - E^*$ is called the "pendant tree" T_v of v.

Reducing pendant trees

First, we remove isolated paths by cutting them into pieces of length ℓ_p. Since the correctness of this is trivial, we omit the proof.

Tree Rule 1 (see Figure 4(a)) *Let p be an isolated path in G. If $|p| \geq \ell_p$, then split off a length-ℓ_p path q and decrease (σ_p, k) by $(1, \omega(q))$. Otherwise, return "NO".*

The next rule cuts branching edges in pendant trees. To this end, it finds an occurrence of a path of length ℓ_p that is furthest from the root of the pendant tree.

Tree Rule 2 (see Figure 4(b)) *Let T_v be the pendant tree of some $v \in V^*$, let u be a leaf of maximal distance to v in T_v. Let W be the set of vertices $\times$ of T_v such that a length-ℓ_p alternating u-x-path exists and let w be a vertex in W maximizing $\mathrm{dist}_{Tv}(v, w)$. Then, delete from G all edges that are incident with the least common ancestor of u and w but are not on the unique u-w-path in T_v.*

Lemma 3 *Let $I = (G, M, \omega, \sigma_p, \sigma_c, \ell_p, \ell_c, k)$ be a yes-instance of* RESTRICTED SCAFFOLDING *such that G is reduced with respect to Tree Rule 2 and let $v \in V^*$. Then, T_v is a path p that is alternating with respect to $M \cap E(T_v)$ and $|p| < \ell_p$ and v is an endpoint of p.*

In the following, we assume that G is reduced with respect to Tree Rule 2 and, thus, we can reject all instances for which Lemma 3 does not hold. Hence, in the following, we assume that Lemma 3 holds for the input instance. The next reduction rule helps unify the way in which pendant trees (which are now paths) attach to G^*, simplifying the rest of the presentation.

Tree Rule 3 (see Figure 4(c)) *Let T_v be the pendant tree of some $v \in V^*$ that is incident with an edge $e \in E(T_v) \setminus M$. Then, delete from G all edges incident with v that are not in $M + e$.*

Reducing long paths

In the following, we assume that G is reduced with respect to the tree reduction rules presented in the previous section. For $i \in \mathbb{N}$, let V_i denote the set of vertices of G that have degree i in G. The goal in this subsection is to reduce the length of chains of degree-two vertices in G. Thus, we consider paths whose inner vertices are all in V_2. We call these paths *deg-2 path*. If the solution has a cycle running through this path, then we cannot modify its length. Therefore, we focus on paths that are guaranteed to not be in a cycle in the solution:

The path reduction rules are based on the idea that the path segments that a deg-2 path p in G is split into by the solution are recurring, that is, if the solution contains the third edge of p, then it also contains the $3 + (\ell_p + 1)^{\text{th}}$ edge of p. This gets slightly more complicated if there are pendants in p, but first, let us consider paths without any pendants.

Path Rule 1 (see Figure 5(a)) *Let $p = (u_0, u_1, \ldots)$ be a deg-2 path with $|p| > \max\{\ell_p, \ell_c\} + \ell_p + 1$ and let $e_i := u_i u_{i+1}$ for all i. Then, for all e_i with $i \leq \ell_p$, add $\omega(e_i)$ to $\omega(e_{\ell_p + i + 1})$ and contract e_i. Finally, decrease σ_p by 1.*

The remaining two rules deal with deg-2 paths containing pendants. Note that any path in the solution that contains the pendant can only "continue" in two directions.

Path Rule 2 (see Figure 5(b)) *Let $p = (u_0, u_1, \ldots, u_{\ell_p + 1} = w)$ and $q = (w = v_0, v_1, \ldots, v_{\ell_p + 1})$ be deg-2 paths and let T_w contain $\gamma > 0$ edges. For all $i \leq \ell_p$, let $e_i := u_i u_{i+1}$, let $f_i := v_i v_{i+1}$ and let $i^+ := (i - \gamma) \bmod (\ell_p + 1)$. Then, for all $i \leq \ell_p$, add $\omega(e_i)$ to $\omega(f_{i^+})$ and contract e_i. Finally, decrease σ_p by 1.*

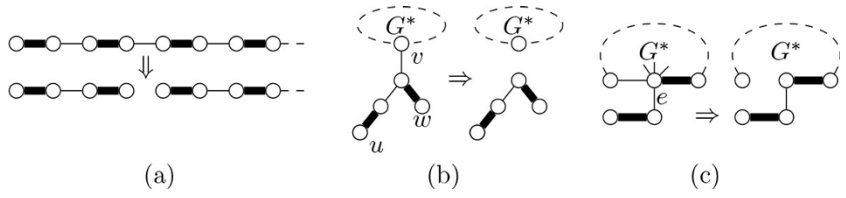

Figure 4 Illustrations for the tree reduction rules.

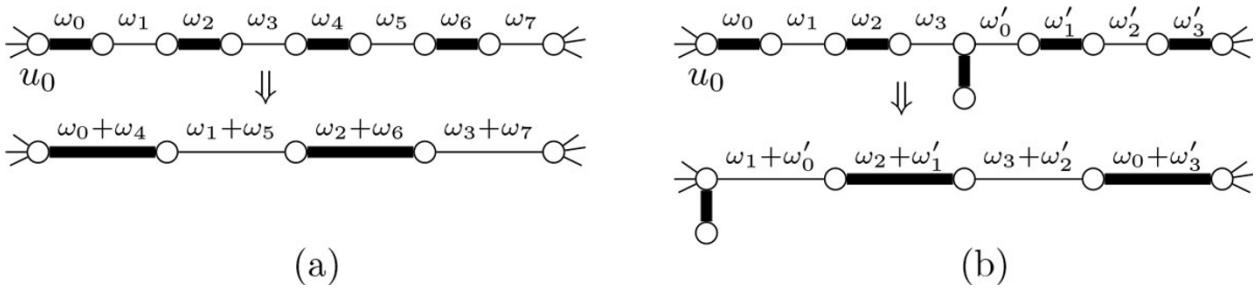

Figure 5 Illustrations for the first path reduction rules.

In case of two pendants, the solution is restricted by the distance between the pendants. If we can infer what the solution does by this length, we implement this right away, otherwise, we can represent the choices that a solution can take with a single pendant.

Path Rule 3 (see Figure 6) *Let $p = (x_0, x_1, \ldots)$ be a u-v-deg-2 path such that $u, v \in V^*$. Let $\gamma_u > 0$ and $\gamma_v > 0$ be the number of edges in T_u and T_v, respectively, and let w be the vertex at maximum distance to u in T_u. Let $\gamma := |p| \bmod (\ell_p + 1)$ and let G' be the result of replacing ux_1 by wx_1 of the same weight in G.*

(1) If $\gamma + \gamma_u \neq_f \ell_p + 1 = \gamma + \gamma_v$, then delete ux_1 (see Figure 6(a)).

(2) If $\gamma + \gamma_u = \ell_p + 1 \neq_f \gamma + \gamma_v$, then delete $vx_{|p|-1}$.

(3) If $\gamma + \gamma_u = \ell_p + 1 = \gamma + \gamma_v$, then return G' (see Figure 6(b)).

(4) If $\gamma = 1$ and $\gamma_u + \gamma_v + 1 = \ell_p$, then return G' (see Figure 6(b)).

(5) If $\gamma = 1$ and $\gamma_u + \gamma_v + 1 \neq \ell_p$, then delete ux_1 (see Figure 6(a)).

(6) If $\gamma \neq 1$ and $\gamma + \gamma_u + \gamma_v \equiv \ell_p \bmod (\ell_p + 1)$, then delete all edges $e \in E(G) \setminus (M \cup \{ux_1\})$.

(7) In all other cases, return "NO".

Finally, we can show that an input graph that is reduced with respect to these rules cannot be larger than $11\ell_p \cdot \mathrm{FES}(G)$ or $11\ell_c \cdot \mathrm{FES}(G)$. To prove this, let $G^\dagger = (V^\dagger, E^\dagger)$ be the result of contracting all degree-2 vertices in G^*.

Lemma 4 *Let G be reduced with respect to all presented reduction rules. Then, $|V| \leq \ell \cdot (|V^\dagger| + 3|E^\dagger|)$ with $\ell := \max\{\ell_c, \ell_p\}$.*

Proof By Lemma 3, we know that $\forall_{v \in V^\dagger} |E(T_v)| < \ell_p$. Therefore, if Lemma 4 is false, there is an edge $uv \in E^\dagger$ such that there are $> 3\ell$ vertices between u and v (i.e. uv is a contraction of more than 3ℓ edges of G). Nevertheless, by irreducibility with respect to Path Rule 3, there is at most one vertex w between u and v such that $T_w$is not empty and the distance between u and v cannot be greater than $2\ell + 1$ (by irreducibility with respect to Path Rule 1 and 2). So, $|E(T_w)| \geq \ell$, contradicting Lemma 3.

Theorem 2 RESTRICTED SCAFFOLDING *admits a kernel containing at most $11\ell \cdot \mathrm{FES}(G)$ vertices and $(11\ell + 1) \cdot \mathrm{FES}(G)$ edges where $\ell := \max\{\ell_p, \ell_c\}$.*

Results

Data

In our experiments, we worked with two datasets, all derived from real genomes (see Table 1). The first one is a set of eleven contig sets, produced from real genomes using the following process: first, the genomes where taken off the NCBI *nucleotide* database (http://www.ncbi.nlm.nih.gov). Then, for each of them, a set of simulated paired-end reads was generated with the tool wgsim ([28]), with default parameters and a 20X mean covering depth. Thereafter, assembly was performed with the tool minia ([29]) with a *k*-mer size $k = 29$. Reads were mapped on the contigs with bwa ([30]), with default parameters and using the sampe method. The second dataset is composed of five scaffold graphs, already presented in [31] and [17]. Some of them have been produced by simulating reads, other come from real paired-end reads libraries (see Table 1). Finally, using the scaftools previously developed

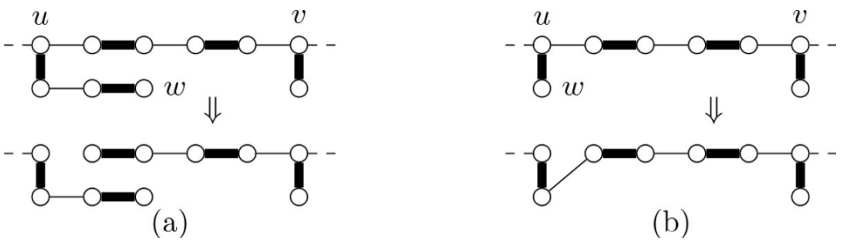

Figure 6 Illustrations for Path Rule 3.

Table 1 Details on the second dataset.

Genome	Reads library	Assembly tool	Mapping tool
staphylo	short jump library (from GAGE [32])	velvet [33]	bwa [30]
ecoli	Illumina reads library SRR001665	velvet	bowtie [34]
ypco92	simulated with wgsim	minia [35]	bwa
wolbachia	simulated with toyseq for the Variathon experiment [36]	minia	bwa
arabido	Illumina reads library SRR616966	velvet	bowtie

in [15,31], we produced the scaffold graphs corresponding to these datasets. See [31] for a more detailed explanation of this pipeline.

The aim of this process is first to produce a benchmark of test graphs that are more realistic than uniformly generated graphs to test our algorithms on, second to study the different parameters that may be interesting to consider for parameterized algorithms, and finally to help to design a more realistic and convenient scaffold graph generator allowing to generate graphs directly, avoiding the complicated pipeline described above (see Table 2 and Table 3). The second dataset was used to study the influence of a preprocessing operation on the scaffold graph, aiming at filtering low informative edges. Simplistically, we removed respectively edges with weight less than 3, 6 and 10. Results are presented in Table 3. We already know that this operation improves the quality of the produced scaffolds, when compared to the original reference genome. Here we would like to observe the behavior of our putative interesting structural parameters according to this filtering.

Graph parameters

Table 2 presents some parameters of the generated graphs. The h-index is the maximum number such that the graph contains h vertices of degree at least h. It measures the degree of connectivity of a graph. The feedback edge set (FES) is the size of a smallest set of edges whose deletion leaves an acyclic graph. The degeneracy is the smallest value d for which every subgraph has a vertex of degree at most d. It is a kind of measure of sparsity of the graph. We notice that scaffold graphs look quite sparse, with few vertices of high degrees and a feedback edge set number that is usually significantly lower than the number of vertices. While degree-based graph parameters like the degeneracy d are tiny in all instances, we recall that our problem generalizes Hamiltonian Cycle, which is already N P-hard on 3-regular graphs. However, Table 2 shows that, for instances that we expect to be seen in practice, these measures can be assumed constant and, thus, it might be worth considering a combination involving these parameters.

Table 3 shows the same statistics for the second set of graphs where edges of small weight (that is, low confidence) were discarded (for weight thresholds of 10, 6, 3 and 0, yielding the original graph with all edges present). We notice that even with a light filtering, the parameters of the scaffold graph are considerably lowered, confirming that raw data suffers from significant noise. Thus, filtering low quality information not only increases the quality of the scaffolding, but also may lead to a significant leap to tractability. Note that, at the time of writing this article, the *Staphylococcus Aureus* genome is no longer available in the NCBI Nucleotide database ("This RefSeq genome was suppressed because updated RefSeq validation criteria identified problems with the assembly or annotation.") and the *Escherichia Coli* genome has been updated since the presented version. Errors in assemblies are quite frequent ([37]) and yield additional issues in scaffolding. Having a better idea of the structure of a classical scaffold graph, and some simple criteria to determine what is anomalous and what is normal (subgraphs induced by repeats for instance) would be of real interest for the analysis of genomes.

Table 2 Scaffold graphs parameters.

Data	$\|V\|$	$\|E\|$	Min/Max/Avg degree	FES	FVS (ub)	tw(ub)	*h*	*dcy*
anopheles^{Ch}	84090	113497	1 / 51 / 2.70	29851	17962		12	3
anthrax^{Ch}	8110	11013	1 / 7 / 2.72	2906	1232	574	7	2
ebola^{Co}	34	43	1 / 5 / 2.53	10	6	3	4	2
gloeobacter^{Ch}	9034	12402	1 / 12 / 2.75	3375	2484	639	8	3
lactobacillus^{Ch}	3796	5233	1 / 12 / 2.76	1439	804	260	8	2
monarch^{Mt}	28	33	1 / 4 / 2.36	6	4	3	4	2
pandora^{Co}	4902	6722	1 / 7 / 2.74	1822	1277	327	7	2
pseudomonas^{Ch}	10496	14334	1 / 9 / 2.73	3851	2692	752	8	2
rice^{Cp}	168	223	1 / 6 / 2.65	56	31	9	5	2
sacchr3^{Ch}	592	823	1 / 7 / 2.78	232	142	43	6	2
sacchr12^{Ch}	1778	2411	1 / 10 / 2.12	637	575	124	7	2

FES = feedback edge set, FVS (ub) = upper bound on the feedback vertex set, *dcy* = degeneracy. Superscripts: Ch = Chromosome, Cp = Chloroplast, Co = Complete, Mt = Mitochondrion

Table 3 Scaffold graph parameters for select genomes and different cut-off thresholds: 0, 3, 6, and 10.

Data & threshold ($\|V\|$)	$\|E\|$	Min/Max/Avg degree	FES	FVS(ub)	tw(ub)	*h*	*dcy*
arabido	0	318984	1 / 31 / 1.85	43593	15703	10610	18	4
(345232)	3	252762	1 / 14 / 1.46	8024	5881	792	9	3
	6	230333	1 / 9 / 1.33	3247	2814	74	8	2
	10	215094	1 / 9 / 1.25	1224	1020	51	8	2
ecoli	0	8142	2 / 46 / 9.40	6411	644	551	32	9
(1732)	3	4043	2 / 23 / 4.67	2312	406	303	16	4
	6	3105	2 / 18 / 3.59	1374	327	162	13	4
	10	2695	2 / 16 / 3.11	964	278	102	11	3
staphylo	0	4765	1 / 128 / 15.83	4164	167	124	52	28
(602)	3	1743	1 / 58 / 5.79	1152	113	57	25	11
	6	1017	1 / 22 / 3.37	464	78	25	14	5
	10	790	1 / 16 / 2.62	279	65	14	11	4
wolbachia	0	1036	1 / 56 / 3.70	481	106	26	11	3
(560)	3	523	1 / 15 / 1.86	47	30	3	6	2
	6	459	1 / 15 / 1.63	19	16	2	5	2
	10	399	1 / 5 / 1.43	12	11	2	5	2
ypco92	0	3465	1 / 8 / 2.61	821	313	191	7	2
(2656)	3	2849	1 / 6 / 2.15	241	136	38	6	2
	6	2651	1 / 5 / 1.99	99	86	3	5	2
	10	2525	1 / 5 / 1.90	73	68	2	5	2

FES = feedback edge set, FVS (ub) = upper bound on the feedback vertex set, tw(ub) = upper bound on the treewidth (by greedy fill-in), *h* = h-index, *dcy* = degeneracy.

Implementation and first results

We implemented the dynamic programming algorithm presented in Section 11 in C++ using boost and the treewidth optimization library TOL [38]. We ran it on a selection of the generated data sets (see Section 11) for which the greedy fill-in heuristic produced tree decompositions of width at most 45. We chose $(\sigma_p, \sigma_c) = (3, 1)$ for the ebola and monarch genomes and $(\sigma_p, \sigma_c) = (20, 3)$ for the more complicated inputs. The tests were run on an AMD Opteron(tm) Processor 6376 at 2300 MHz.

Figure 7 shows running times and memory consumption needed to produce a solution as well as details regarding the used tree decompositions. Figure 8 shows optimal solutions for the two smallest data-sets: ebola and monarch. To validate the proposed scaffolding, we compared the output to the alignment of the contigs on the reference sequence using megablast ([39]). In the case of the ebola genome, the two isolated contigs (in green) are small (about 150 bp). One of them is placed between the orange contig and its neighbor, the other one finds its place at the other extremity of the chain. In the input scaffold graph, we notice that they are linked to the wrong node, we suppose this is due to their small size, disturbing the alignment step. For the monarch mitochondrion, however, two of the contigs (appearing in red) did not match on the sequence, meaning that the assembly yields some errors. The isolated contig is also small and not correctly connected in the scaffold graph.

data set	treewidth	#bags	bag-size distribution	t[s]	mem[MB]
monarch	2	23	2 3	0.00	2.4
ebola	3	31	2 3 4	0.00	2.5
rice	9	157	2 3 4 5 6 7 8 9 10	1.76	63
wolbachia (cut-off 3)	3	440	2 3 4	0.29	18
wolbachia (cut-off 6)	2	420	2 3	0.12	18
wolbachia (cut-off 10)	2	375	2 3	0.09	18
ypco92 (cut-off 6)	3	2411	2 3 4	1.24	340
ypco92 (cut-off 10)	3	2366	2 3	1.19	340

Figure 7 Information on the tree decomposition and resource requirements of running the algorithm for select instances with small treewidth. We chose $(\sigma_p, \sigma_c) = (3, 1)$ for ebola and monarch, (20, 3) for rice and (256, 16) for the second data set (with edge cut-offs).

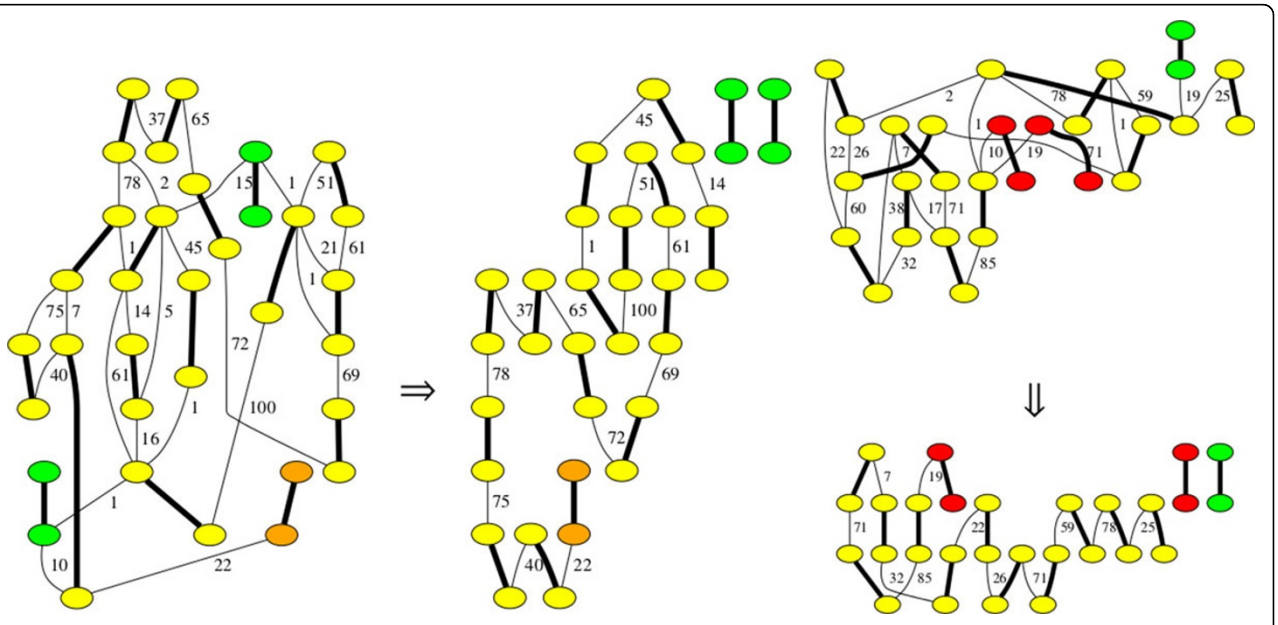

Figure 8 Optimal solutions for the ebola (left) and monarch (right) graphs. Edges have their weights on the right. Matching edges are bold and normalized to weight 0.

Concerning the rice chloroplast, among the 84 contigs, only three were misplaced. All three of them are small (< 130 bp), two of them strongly overlap and the third has two occurrences in the reference genome, one complete and one partial. The seven remaining scaffolds follow exactly the right relative order and orientation of the contigs on the reference genome. Chloroplast genomes have a particularity which make them interesting as data for scaffolding. They present an inverted repeat region of approximately 20 kbp [40]. Figure 9 focuses on one of the scaffolds, where this inverted repeat is shown in pink. The other occurrence is not present in the scaffolding. To notice, the weights inside this repeat are in average higher than outside, which is totally expected since the read cover is approximately doubled for these sequences. Thus, areas of the graph with higher weight would lead to repeat hypothesis, if we are confident in the homogeneity of the cover.

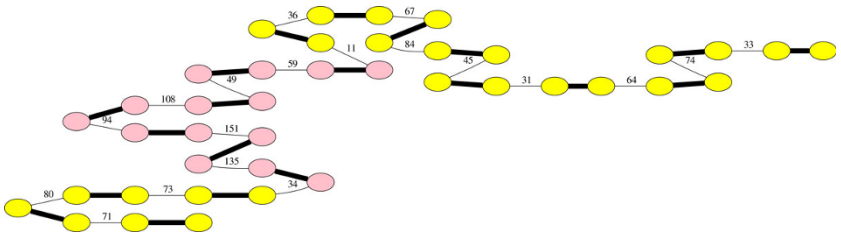

Figure 9 Scaffold in the rice chloroplast genome, including the inverted repeat.

Discussion and conclusion

In this paper, we considered exact approaches to the $\mathcal{NP}$-hard SCAFFOLDING problem which is an integral part of genome sequencing. We showed that it can be solved in polynomial time on trees or graphs that are close to being trees (constant treewidth) by a dynamic programming algorithm. We proved a linear-size problem kernel for the parameter "feedback edge set" for a restricted version in which the lengths of paths and cycles are fixed. We implemented an exact algorithm solving SCAFFOLDING in $f(\text{tw}) \cdot \text{poly}(n)$ time and evaluated our implementation experimentally, supporting the claim that this method produces high-quality scaffolds. Our experiments are run on data sets that are based on specific real-world genomes, which we also examined to identify a number of interesting parameters that may help design parameterized algorithms and random scaffold graph generators that produce more realistic instances. We are currently transferring the preprocessing rules to the general problem variant. We are highly interested in further graph classes that are closer to real-world instances than trees and on which the problem might be polynomial-time solvable. From an algorithmic point of view, we remark that only few bags of the used tree decompositions are small (see Figure 7). Thus, we envision a hybrid strategy of branching on the vertices in the largest bags before running the dynamic-programming algorithm and using this "distance x to tree-width-x" parameter to re-analyze the problem.

We intend to perform more extensive tests on diverse datasets, in particular comparing the quality of this approach to existing ones using, for instance, the criteria presented in [9]. This work demands reliable benchmarks, with up-to-date and well assembled genomes. From a bioinformatics point of view, we lay the groundwork for a careful analysis of the scaffold graph, as well as a tool to speed up the above algorithms, as a help to analyze the quality of the assembly, and maybe the structure of the genome itself.

Appendix

Proof of Lemma 1 The proof is by induction over the distance of v to r (descending) and, in case of ties, j (ascending).

Induction Base: The statement holds for all vertices v if $j = 0$ since $T_0^v = T[\{v\}]$ does not contain edges.

Induction Step (≥): First, we show that $[c, i, j]_v \geq \omega(S)$. To this end, let $u := s_v[j]$ and, noting that all vertices except maybe v of T_j^v are incident with edges in M, let q denote the path in $G[S \cup M]$ containing u. Let w be the vertex paired with v by M. Let α denote the number of edges in $q \cap E(T_u + v) \setminus M$ incident with u and let $\beta \leq c$ denote the number of edges in $q \cap E(T_{j-1}^v) \setminus M$ incident with $s_v[1..(j-1)]$. Let $S_u := \bigcup_{p \in S} p \cap E(T_u)$ and let $S_{j-1} := \bigcup_{p \in S} p \cap E(T_{j-1}^v)$ and note that S_u and S_{j-1} are path covers of T_u and T_{j-1}^v, respectively. Thus, by induction hypothesis,

$$\omega(S_u) \leq [\alpha, \| S_u \|_p, \infty]_u \text{ and } \omega(S_{j-1}) \leq [\beta, \| S_{j-1} \|_p, j-1]_v. \quad (1)$$

Case 1: $uv \in M$ and $c = 0$. Then, $\| S \|_p = \| S_u \|_p^{T_u + v} + \| S_{j-1} \|_p - \| S_u \|_p + (1 - \alpha) + \| S_{j-1} \|_p$ and, thus,

$$[c, i, j]_v \overset{line\,6}{\geq} [\alpha, \| S_u \|_p, \infty]_u + [0, i - (\| S_u \|_p - \alpha + 1), j-1]_v \overset{(1)}{\geq} \omega(S_u) + \omega(S_{j-1}) = \omega(S).$$

Case 2: $uv \in M$ and $c = 1$. Then, $\beta = 1$. Furthermore, the path containing u is split over $S_u + uv$ and S_{j-1}, implying $\| S \|_p = \| S_u \|_p^{T_u + v} + \| S_{j-1} \|_p - 1 = \| S_u \|_p - \alpha + \| S_{j-1} \|_p$. Thus,

$$[c, i, j]_v \overset{line\,7}{\geq} [\alpha, \| S_u \|_p, \infty]_u + [0, i - (\| S_u \|_p - \alpha), j-1]_v \overset{(1)}{\geq} \omega(S_u) + \omega(S_{j-1}) = \omega(S).$$

Case 3: $uv \notin M$ and $c = 0$. Then, $||S||_p = ||S_u||_p + ||S_{j-1}||_p$ and we have

$$[c, i, j]_v \overset{line\,9}{\geq} [\alpha, \| S_u \|_p, \infty]_u + [0, i - \| S_u \|_p, j-1]_v \overset{(1)}{\geq} \omega(S_u) + \omega(S_{j-1}) = \omega(S).$$

Case 4: $uv \notin M$ and $c = 1$. If $uv \notin S$, then $\beta = 1$. Furthermore, $||S||_p = ||S_u||_p + ||S_{j-1}||_p$ and we have

$$[c, i, j]_v \overset{line\,10}{\geq} [\alpha, \| S_u \|_p, \infty]_u + [1, i - \| S_u \|_p, j-1]_v \overset{(1)}{\geq} \omega(S_u) + \omega(S_{j-1}) = \omega(S).$$

Otherwise, $uv \in S$, implying $\alpha = \beta = 0$. If $w \notin s_v[1..j]$, then $\| S \|_p = \| S_u + uv \|_p^{T_u + v} + \| S_{j-1} \|_p = \| S_u \|_p + \| S_{j-1} \|_p$ and

$$[c, i, j]_v \overset{line\,10}{\geq} \omega(uv) + [0, \| S_u \|_p, \infty]_u + [0, i - \| S_u \|_p, j-1]_v \overset{(1)}{\geq} \omega(uv) + \omega(S_u) + \omega(S_{j-1}) = \omega(S).$$

Otherwise, $w \in s_v[1..j]$ and the path containing u is split over S_u and S_{j-1}. Thus, $\| S\|_p = \| S_u + uv \|_p^{T_u+v} + \| S_{j-1}\|_p - 1 = \| S_u\|_p + \| S_{j-1}\|_p - 1$, implying

$$[c, i, j]_v \overset{\text{line10}}{\geq} \omega(uv) + [0, \| S_u\|_p, \infty]_u + [0, i - (\| S_u\|_p - 1), j - 1]_v \overset{(1)}{\geq} \omega(uv) + \omega(S_u) + \omega(S_{j-1}) = \omega(S).$$

Induction Step (≤): Next, we show that $[c, i, j]_v \leq \omega(S)$ by proving that a i-path cover S' for T_j^v of weight $[c, i, j]_v$ exists and, thus, $[c, i, j]_v = \omega(S') \leq \omega(S)$ by optimality of S. To this end, let $u := s_v[j]$ and let $w := M(v)$.

Case 1: $w = u$ and $c = 0$. By line 6, there are ℓ and α such that $[0, i, j]_v = [\alpha, \ell, \infty]_u + [0, i - (\ell - \alpha + 1), j - 1]_v$. By induction hypothesis, there are path covers S_u and S_{j-1} corresponding to $[\alpha, \ell, \infty]_u$ and $[0, i - (\ell - \alpha + 1), j - 1]_v$, respectively. Then, $S' := S_u \uplus S_{j-1}$ is a path cover for T_j^v and $\| S'\|_p = \| S_u\|_p^{T_u+v} + \| S_{j-1}\|_p = \| S_u\|_p + 1 - \alpha + \| S_{j-1}\|_p = i$. Furthermore,

$$\omega(S') = \omega(S_u) + \omega(S_{j-1}) \overset{\text{ind.hyp.}}{=} [\alpha, \ell, \infty]_u + [0, i - (\ell - \alpha + 1), j - 1]_v \overset{\text{line 6}}{=} [0, i, j]_v.$$

Case 2: $u = w$ and $c = 1$. By line 7, there are ℓ and α such that $[1, i, j]_v = [\alpha, \ell, \infty]_u + [1, i - (\ell - \alpha), j - 1]_v$. By induction hypothesis, there are path covers S_u and S_{j-1} corresponding to $[\alpha, \ell, \infty]_u$ and $[1, i - (\ell - \alpha), j - 1]_v$, respectively. Then, $S' := S_u \cup S_{j-1}$ is a path cover for T_j^v and $\| S'\|_p = \| S_u\|_p^{T_u+v} + \| S_{j-1}\|_p - 1$ since S_{j-1} contains an edge incident to v. Thus, $\| S'\|_p = \| S_u\|_p + \| S_{j-1}\|_p - \alpha = i$. Furthermore,

$$\omega(S') = \omega(S_u) + \omega(S_{j-1}) \overset{\text{ind.hyp.}}{=} [\alpha, \ell, \infty]_u + [1, i - (\ell - \alpha), j - 1]_v \overset{\text{line7}}{=} [1, i, j]_v.$$

Case 3: $w \neq u$ and $c = 0$. By line 9, there are ℓ and α such that $[0, i, j]_v = [\alpha, \ell, \infty]_u + [0, i - \ell, j - 1]_v$. By induction hypothesis, there are path covers S_u and S_{j-1} corresponding to $[\alpha, \ell, \infty]_u$ and $[0, i - \ell, j - 1]_v$, respectively. Since $c = 0$ we have $uv \notin S_u$ and, thus, $S' := S_u \uplus S_{j-1}$ is a path cover for T_j^v and $\| S'\|_p = \| S_u\|_p + \| S_{j-1}\|_p = i$. Furthermore,

$$\omega(S') = \omega(S_u) + \omega(S_{j-1}) \overset{\text{ind.hyp.}}{=} [\alpha, \ell, \infty]_u + [0, i - \ell, j - 1]_v \overset{\text{line9}}{=} [0, i, j]_v.$$

Case 4: $w \neq u$ and $c = 1$.

Case 4a: There are ℓ and α such that $[1, i, j]_v = [\alpha, \ell, \infty]_u + [1, i - \ell, j - 1]_v$ (see line 10). By induction hypothesis, there are path covers S_u and S_{j-1} corresponding to $[\alpha, \ell, \infty]_u$ and $[1, i - \ell + 1, j - 1]_v$, respectively. Since $uv \notin M$, some edge incident to u in T_u is in M. Then, $S' := S_u \uplus S_{j-1}$ is a path cover for T_j^v and $||S'||_p = ||S_u||_p + ||S_{j-1}||_p = i$. Furthermore,

$$\omega(S') = \omega(S_u) + \omega(S_{j-1}) \overset{\text{ind.hyp.}}{=} [\alpha, \ell, \infty]_u + [1, i - \ell, j - 1]_v \overset{\text{line10}}{=} [1, i, j]_v.$$

Case 4b: There is some ℓ such that $[1, i, j]_v = \omega(uv) + [0, \ell, \infty]_u + [1, i - (\ell - 1), j - 1]_v$ (see line 10). Then, $w \in s_v[1..j]$ and, by induction hypothesis, there are path covers S_u and S_{j-1} corresponding to $[0, \ell, \infty]_u$ and $[1, i-\ell+1, j-1]_v$, respectively. Then, $S' := (S_u + uv) \cup S_{j-1}$ is a path cover for T_j^v and w, v and u are on the same path p in $T_j^v[S' \cup M]$. Since u is incident to an edge of M in T_u, we know that p does not end in u. Thus, $||S'||_p = ||S_u||_p + ||S_{j-1}||_p - 1 = i$. Furthermore,

$$\omega(S') = \omega(S_u) + \omega(S_{j-1}) + \omega(uv) \overset{\text{ind.hyp.}}{=} [0, \ell, \infty]_u + [1, i - (\ell + 1), j - 1]_v \overset{\text{line10}}{=} [1, i, j]_v.$$

Case 4c: There is some ℓ such that $[1, i, j]_v = \omega(uv) + [0, \ell, \infty]_u + [1, i - \ell, j - 1]_v$ (see line 10). Then, $w \notin s_v[1..j]$ and, by induction hypothesis, there are path covers S_u and S_{j-1} corresponding to $[0, \ell, \infty]_u$ and $[1, i - \ell, j - 1]_v$, respectively. Thus, $S' := (S_u + uv) \cup S_{j-1}$ is a path cover for T_j^v. Since u is incident to an edge of M in T_u, we have $||S'||_p = ||S_u||_p + ||S_{j-1}||_p = i$. Furthermore,

$$\omega(S') = \omega(S_u) + \omega(S_{j-1}) + \omega(uv) \overset{\text{ind.hyp.}}{=} [0, \ell, \infty]_u + [1, i - \ell, j - 1]_v \overset{\text{line10}}{=} [1, i, j]_v.$$

Proof of Section 11

Proof of Lemma 2 The proof is by induction on the distance of i to r (descending). In the induction base, i is a leaf of T and $X(i) = \emptyset$ and G_i is empty. Thus, the domains of d and P are empty. Thus, $[\emptyset, \emptyset, 0, 0]_i = 0$ and all other entries are $-\infty$.

For the induction step, we distinguish the possible bag types of $X(i)$ with children $X(j)$ and $X(\ell)$ (possibly $j = \ell$):

Introduce vertex v: Since G_i does not contain edges incident to v, only tuples with $d(v) = 0$ and $P(v) = \perp$ are valid.

Forget vertex v: Let S_i be a maximum weight set that is eligible for (d, P, p, c, i). We show that $[d, P, p, c]_i = \omega(S_i)$.

"≤":

Case 1: $[d, P, p, c]_i = [d[v \to 1], P + vv, p - 1, c]_j$. By induction hypothesis, there is a set S_j corresponding[1] to $[d[v \to 1], P + vv, p - 1, c]_j$. We show that S_j is eligible for (d, P, p, c, i) and, thus, $\omega(S_i) \geq \omega(S_j) = [d, P, p, c]_i$. Since $X(i) \subset X(j)$ and all paths between vertices in $d^{-1}(1)$ that are represented by $P + vv$ are also represented by P, the first two conditions are satisfied by S_j for i. Since $\deg_{G_j^{S_j}}(v) = 1$, there is a path q in $G_j^{S_j}$ containing v. But since $v \notin d^{-1}(1)$, we know that $G_i^{S_i}$ contains one path more that does not intersect $d^{-1}(1)$. Thus, $G_i^{S_i}$ contains p paths that do not intersect $d^{-1}(1)$ and S_j satisfies the third condition.

Case 2: $[d, P, p, c]_i = [d[v \to 1], (P - uu) + uv, p, c]_j$ for some $uu \in P$. By induction hypothesis, there is a set S_j corresponding to $[d[v \to 1], (P - uu) + uv, p, c]_j$. Since $uv \in (P - uu) + uv$, there is a u-v-path q in $G_j^{S_j}$ (by Definition 2(2)) and q intersects $d^{-1}(1)$ in u.

Thus, $G_i^{S_i}$ contains p paths that do not intersect $d^{-1}(1)$ and, thus, S_j is eligible for (d, P, p, c, i).

Case 3: $[d, P, p, c]_i = [d[v \to x], P, p, c]_j$ for some $x \in \{0, 2\}$. By induction hypothesis, there is a set S_j corresponding to $[d[v \to x], P, p, c]_j$. Then, S_j is also eligible for (d, P, p, c, i).

"≥": Let $x := \deg_{G_i^{S_i}}(v)$.

Case 1: $x \in \{0, 2\}$. Then, S_i is eligible for $(d[v \to x], P, p, c, j)$ and, by induction hypothesis, $\omega(S_i) \le [d[v \to x], P, p, c]_j \le [d, P, p, c]_i$.

Case 2: $x = 1$. Then, by Definition 2(1), there is a path q in $G_i^{S_i}$ ending in v. If q has another end u in $d^{-1}(1)$, then S_i is eligible for $(d[v \to 1], (P - uu) + uv, p - 1, c, j)$ and, thus, $\omega(S_i) \le [d[v \to 1], (P - uu) + uv, p - 1, c]$. Otherwise, S_i is eligible for $(d[v \to 1], P + vv, p - 1, c, j)$. In both cases, $\omega(S_i) \le [d, P, p, c]_i$.

Introduce edge uv**:** Let S_i correspond to $[d, P, p, c]_i$. We show that $[d, P, p, c]_i = \omega(S_i)$. Let d' and z be as described in the dynamic programming and note that $G_j^{=} G_i^{-} uv$. "≤": First, if $[d, P, p, c]_i = [d, P, p, c]_j$, then, by induction hypothesis, there is a set S_j corresponding to $[d, P, p, c]_j$ and $uv \notin M$. Then, $G_j^{S_j} = G_i^{S_j}$ implying that S_j is eligible for (d, P, p, c, i) and, thus, $\omega(S_i) \ge \omega(S_j) = [d, P, p, c]_i$.

In the following, we proceed in a similar manner for the case that $[d, P, p, c]_i = z + \omega(uv)$: To show $[d, P, p, c]_i \le \omega(S_i)$, we consider a set S_j that corresponds to the entry for $X(j)$ from which $[d, P, p, c]_i$ is computed and whose existence is granted by induction hypothesis. Then, we show that $S_j + uv$ is eligible for (d, P, p, c, i), implying $\omega(S_i) \ge \omega(S_j + uv) = z + \omega(uv)$ if $uv \notin M$ and $\omega(S_i) \ge \omega(S_j) = z$ if $uv \in M$. Note that, in each of the cases, the degrees of u and v in $G_j^{S_j}$ are one less than their degrees in $G_i^{S_j+uv}$. Thus, S_j satisfies Definition 2(1) for d'.

[1]We say a set S *corresponds* to an entry $[d, P, p, c]_i$ if S is eligible for (d, P, p, c, i) and its weight $\omega(S) = [d, P, p, c]_i$is maximum among all sets that are.

Case 1: $d(u) = d(v) = 2$. Then, both u and v have degree 1 in $G_j^{S_j}$.

Case 1a: $z = [d', P + uv, p, c - 1]_j$. Then, there is a u-v-path in $G_j^{S_j}$ (see Figure 3(a)). Since $d^{-1}(1) = d'^{-1}(1) \setminus \{u, v\}$, adding uv does not touch any paths intersecting $d^{-1}(1)$. Thus, Definition 2(2) is satisfied. Further, adding uv closes a cycle in $G_i^{S_j+uv}$ and, thus, $G_i^{S_j+uv}$ contains one more cycle that does not intersect $d^{-1}(1)$ than $G_j^{S_j}$. Thus, also Definition 2(3) is satisfied.

Case 1b: $z = [d', P \cup \{uu, vv\}, p - 1, c]_j$. Then, both u and v have dangling paths in $G_j^{S_j}$ (see Figure 3(b)). By the same arguments as in Case 1a, Definition 2(2) is satisfied. Further, adding uv connects two paths such that the resulting path does not intersect $d^{-1}(1)$. Thus, there are one more such paths in $G_i^{S_j+uv}$ than in $G_j^{S_j}$, implying that Definition 2(3) is satisfied.

Case 1c: $z = [d', (P - xx) \cup \{ux, vv\}, p, c]_j$ for some $x \in d^{-1}(1) = d'^{-1}(1) \setminus \{u, v\}$ with $xx \in P$. Then, $G_j^{S_j}$ contains a path dangling from v and a u-x-path (see Figure 3(c)). Thus, adding uv connects two paths such that the resulting path intersects $d^{-1}(1)$ exclusively in x. Hence, there is a path dangling from x in $G_i^{S_j+uv}$ and, hence, $S_j + uv$ satisfies Definition 2(2). Since no paths or cycles avoiding $d'^{-1}(1)$ are affected by adding uv, Definition 2(3) is satisfied. The case that $z = [d', (P - xx) \cup \{vx, uu\}, p, c]_j$ is analogous.

Case 1d: $z = [d', (P - xy) \cup \{ux, vy\}, p, c]_j$ for some $x, y \in d^{-1}(1)$ with $xy \in P$. Then, $G_j^{S_j}$ contains paths q_u and q_v that start in u and v, respectively, and end in x and y, respectively (see Figure 3(d)). Thus, adding uv connects q_u and q_v to a single path q that starts in x and ends in y, thus intersecting $d^{-1}(1)$. Hence both Definition 2(2) and (3) are satisfied.

Case 2: $d(u) = d(v) = 1$. Thus, u and v are not incident to any edges in $G_j^{S_j}$. Thus, uv forms a new path connecting u and v in $G_i^{S_j+uv}$. Since, in this case, $z = [d', P - uv, p, c]_j$and $uv \in P$, we conclude that both Definition 2(2) and (3) are satisfied.

Case 3: $d(u) = 2$, $d(v) = 1$. Then, v has no incident edges in $G_j^{S_j}$ and, thus, it is only adjacent to u in $G_i^{S_j+uv}$. Further, u has degree 1 in $G_j^{S_j}$, so it is endpoint to a path q in $G_j^{S_j}$. Thus, $vx \in P$ for some $x \in d^{-1}(1)$ and $G_i^{S_j+uv}$ contains the path $q' := q + uv$.

Case 3a: $x = v$. Then, $z = [d', (P - vv) + uu, p, c]_j$. Since $G_i^{S_j+uv}$ contains q' which is a path dangling from u, Definition 2(2) is satisfied. Since no other paths are touched, Definition 2(3) is satisfied.

Case 3b: $x \neq v$. Then, $z = [d', (P - vx) + ux, p, c]_j$. Since $G_i^{S_j+uv}$ contains q' which is a u-x-path, Definition 2(2) is satisfied. Since no other paths are touched, Definition 2(3) is satisfied.

"≥": If $uv \notin S_i \cap M$, then $G_i^{S_i} = G_j^{S_j}$ and, thus, S_i is eligible for (d, P, p, c, j). By induction hypothesis, $[d, P, p, c]_j \ge \omega(S_i)$ and, thus, $[d, P, p, c]_i \ge [d, P, p, c]_j\} \ge \omega(S_i)$. Otherwise, $uv \in S_i \cup M$

In the following, we show that $S_i - uv$ is eligible for a tuple corresponding to one of the entries over which we maximize to compute z. Thus, $\omega(S_i) \le \omega(S_i - uv) + \omega(uv) \le z + \omega(uv)$ if uv tt M and $\omega(S_i) \le z$ if $uv \in M$.

Note that, in each case, $S_i - uv$ satisfies Definition 2(1) since the degrees of u and v decrease by one when removing uv.

Case 1: $d(u) = d(v) = 2$. Then, uv is part of a path q in $G_i^{S_i}$ and neither u nor v is an endpoint of q.

Case 1a: q is closed (that is, a cycle) (see Figure 3(a)). Then, q does not intersect $d^{-1}(1)$, implying that $G_i^{S_i}$ contains one more such cycle than $G_j^{S_i-uv}$. Further, $q - uv$ is a u-v-path in $G_j^{S_i-uv}$ intersecting $d'^{-1}(1)$ only in u and v. Thus, $S_i - uv$ is eligible for $(d', P + uv, p, c - 1, j)$.

Case 1b: q is open and does not intersect $d^{-1}(1)$ (see Figure 3(b)). Then, $G_i^{S_i}$ contains one more of such paths than $G_j^{S_i-uv}$. Further, $q - uv$ decomposes into paths q_u and $q_v$intersecting $d'^{-1}(1)$ only in u and v, respectively. Thus, $S_i - uv$ is eligible for $(d', P + \{uu, vv\}, p - 1, c, j)$.

Case 1c: q is open and intersects $d^{-1}(1)$ in a single vertex x (see Figure 3(c)). Then, $q - uv$ decomposes into paths q_u and q_v in $G_j^{S_i-uv}$, one of which intersects $d'^{-1}(1)$ in x. Since no path avoiding $d^{-1}(1)$ is touched, $S_i - uv$ is eligible for either $(d', (P - xx) \cup \{xu, vv\}, p, c, j)$ or $(d', (P - xx) \cup \{uu, vx\}, p, c, j)$.

Case 1d: q is open and intersects $d^{-1}(1)$ in 2 distinct vertices x and y (see Figure 3(d)). Then, $q - uv$ decomposes into a u-x-path q_u and a v-y-path q_v in $G_j^{S_i-uv}$. Thus, S_i - uv is eligible for $(d', (P - xy) \cup \{ux, vy\}, p, c, j)$.

Case 2: $d(u) = d(v) = 1$. Then, $q = \{uv\}$ is a path in $G_i^{S_i}$ and, thus, u and v are isolated in $G_j^{S_i-uv}$. Thus, $uv \in P$ and $S_i - uv$ is eligible for $(d', P - uv, p, c, j)$.

Case 3: $d(u) = 2$ and $d(v) = 1$. Thus, $G_i^{S_i}$ contains a path $q = (v, u, \ldots)$. If q intersects $d^{-1}(1) - v$ in a vertex x, then $q - uv$ intersects $d'^{-1}(1) - u$ and, thus, $S_i - uv$ is eligible for $(d', (P - vx) + ux, p, c, j)$. Otherwise, q is a dangling from v in $G_i^{S_i}$ and $q - uv$ is a dangling from u in $G_j^{S_i-uv}$, implying that $S_i - uv$ is eligible for $(d', (P - vv) + uu, p, c, j)$.

Join: "≤": Let d_1, d_2, P_1, P_2, p_1, p_2, c_1, c_2 be such that $[d, P, p, c]_i = [d_1, P_1, p_1, c_1]_j + [d_2, P_2, p_2, c_2]_\ell$. By induction hypothesis, there are sets S_j and S_ℓ corresponding to $[d_1, P_1, p_1, c_1]_j$ and $[d_2, P_2, p_2, c_2]_\ell$, respectively, such that

$$\forall_{v \in X(i)}\ d(v) = d_1(v) + d_2(v)\ , \tag{2}$$

$$P = P_1 \sqcup P_2, \tag{3}$$

$$p = p_1 + p_2 + |(P_1 \cap P_2)^1|, \text{ and} \tag{4}$$

$$c = c_1 + c_2 + |(P_1 \cap P_2)^2|. \tag{5}$$

We show that $S_i := S_j \cup S_\ell$ is eligible for (d, P, p, c, i) and, thus, $\omega(S) \geq \omega(S_i) = [d_1, P_1, p_1, c_1]_j + [d_2, P_2, p_2, c_2]_\ell = [d, P, p, c]_i$. First, by (2), we have that Definition 2(1) is satisfied. Second, to show that Definition 2(2) is satisfied, consider some $uv \in P$. Then, there is a path $q = (u, x_1, x_2, \ldots, v)$ in $\mathrm{Gr}(P_1 \cup P_2)$. Note that $x_1, x_2, \ldots \in d_1^{-1}(1) \cup d_2^{-1}(1)$. Thus, $G_i^{S_j \cup S_\ell} = G_i^{S_i}$ contains a u-x_1-path, an x_1-x_2-path, ... The concatenation of these paths forms a u-v path in $G_i^{S_i}$. Third, note that, for each $uu \in (P_1)^1$, there is a path q_1^u dangling from u in $G_j^{S_j}$ such that the only vertex of $d_1^{-1}(1) \cup d_1^{-1}(0) \supseteq d^{-1}(1)$ in q_1^u is u. Analogously, a similar path q_2^u exists for each $uu \in (P_2)^1$ in $G_\ell^{S_\ell}$. Thus, for each $uu \in (P_1 \cap P_2)^1$, there is a path $q^u := q_1^u \cup q_2^u$ in $G_i^{S_j \cup S_\ell}$ containing u and avoiding $d^{-1}(1)$ and q^u is neither in $G_j^{S_j}$ nor in $G_\ell^{S_\ell}$. Since $G_j^{S_j}$ contains p_1 paths avoiding $d_1^{-1}(1) \cup d_1^{-1}(0) \supseteq d^{-1}(1)$ and $G_\ell^{S_\ell}$ contains p_2 paths avoiding $d_2^{-1}(1) \cup d_2^{-1}(0) \supseteq d^{-1}(1)$, we have that $G_i^{S_j \cup S_\ell}$ contains $p_1 + p_2 + |(P_1 \cap P_2)^1|$ such paths. Similarly, $G_i^{S_j \cup S_\ell}$ concontains $c_1 + c_2 + |(P_1 \cap P_2)^2|$ cycles avoiding $d^{-1}(1)$.

"≥": Let $S_j := S_i \cap E(G_j)$ and let $S_\ell := S_i \cap E(G_\ell)$. We show that S_j and $S_\ell$are eligible for tuples (d_1, P_1, p_1, c_1, j) and $(d_2, P_2, p_2, c_2, \ell)$, respectively, such that (2)-(5) hold. First, for all $u \in X(i) = X(j) = X(\ell)$ let d_1 (d_2) be the number of edges of G_j (G_ℓ) incident with u. Since $E(G_i^{S_i}) = E(G_j^{S_j}) \uplus E(G_\ell^{S_\ell})$, we conclude that (2) holds and Definition 2(1) is satisfied. Second, let P_1 be the set of pairs uv with $u, v \in d_1{}^{-1}(1)$ such that, if $u \neq v$, then there is a u-v path in $G_j^{S_j}$ and, if $u = v$, then $G_j^{S_j}$ contains a path dangling from u. Let P_2 be defined analogously for d_2 and $G_\ell^{S_\ell}$. Then, Definition 2(2) is satisfied. Further, for all $uv \in P$, since $d(u) = 1 \Longleftrightarrow d_1(u) = 1 \oplus d_2(u) = 1$, we have $P_1(u) = \perp \oplus P_2(u) = \perp$ and there is a u-v-path q in $G_i^{S_i}$ that decomposes into paths of $G_j^{S_j}$ that connect vertices of $d_1{}^{-1}(1)$ and paths of $G_\ell^{S_\ell}$ that connect vertices of $d_2{}^{-1}(1)$. Thus, $\mathrm{Gr}(P_1 \cup P_2)$ contains a u-v-path and we conclude that (3) holds. Third, let p_1 and c_1 be the number of paths and cycles, respectively, in $G_j^{S_j}$ that avoid $d_1{}^{-1}(1)$. Likewise for p_2 and c_2 in $G_\ell^{S_\ell}$ and d_2. Then Definition 2(3) is satisfied. Further, let p' denote the number of pairs $uu \in P_1 \cap P_2$, that is, $p' := |(P_1 \cap P_2)^1|$. Then,

since $G_i^{S_j \cup S_\ell}$ contains, for each such pair uu, a different path consisting of the concatenation of the two paths dangling from u in $G_j^{S_j}$ and $G_\ell^{S_\ell}$, respectively, $G_i^{S_j \cup S_\ell}$ contains exactly $p_1 + p_2 + p'$ paths avoiding $d^{-1}(1)$. Thus, (4) holds and, in complete analogy, (5) holds. □

Proof of Theorem 1, sketch The bottleneck in the computation are the join nodes so we focus on computing their dynamic programming table. To calculate the number of entries that have to be considered in order to compute $[d, P, p, c]_i$, assume that P_1 and P_2 are fixed. Then so is P and ${d_1}^{-1}(1)$ and ${d_2}^{-1}(1)$. Now, consider a vertex $u \in X(j) \backslash d_1^{-1}(1)$. If u is incident to an edge in $G_j^{\oslash}$, that is, an edge in M that has been introduced in the subtree rooted at j, then $d_1(u) > 0$ and, thus, $d_1(u) = 2$. However, if u is not incident with any matching edges in $G_j^{\oslash}$, then $d_1(u) < 2$, since otherwise, u would be incident to two non-matching edges. Thus, $u \in d_1^{-1}(2)$ if and only if u is incident to a matching edge in $G_j^{\oslash}$ and, consequently, fixing P_1 and P_2 also fixes d_1 and, by extension, d_2. Finally, we can choose p_1 and c_1 in order to compute p_2 and c_2. Since we need to consider only permutations that are also involutions, there are less than $\mathrm{tw}^{\mathrm{tw}}$ ways to choose P_1 and P_2. Thus, the maximum is over at most $\mathrm{tw}^{\mathrm{tw}} \cdot \sigma_p \cdot \sigma_c$ elements. Since there are $O(n)$ bags in the tree decomposition, the algorithm can be executed in the claimed running time. □

Lemma 5 *Tree Rule 2 is correct, that is, the instance* $I = (G, M, \sigma_p, \sigma_c, \ell_p, \ell_c)$ *is yes if and only if the result* $I' = (G', M', \sigma_p - 1, \sigma_c, \ell_p, \ell_c)$ *of applying Tree Rule 2 to I is yes.*

Proof Clearly, since all vertices of the graph G must be covered, then the only way to pack the vertex u is to include it into a path of length l_p included the vertex $v' = LCA(u, w)$ and then decrease the number of paths by one. □

Proof of Lemma 3 We show that T_v does not contain branching vertices (vertices with at least two children) since it is straightforward that, if T_v is a path, it has to be alternating for I to be a yes-instance and, if its length exceeds ℓ_p, then Tree Rule 2 applies. Towards a contradiction, assume that T_v has branching vertices and let z denote such a vertex in T_v such that, among all branching vertices, z is furthest from v. Let u be a leaf of T_z that has maximum distance to z and let d denote this distance. Since z is the only branching vertex in T_z and I is a yes-instance, the unique u-z path in T_z is alternating. Thus, by irreducibility with respect to Tree Rule 2, we know that $d < \ell_p$. However, since z is branching, there is another leaf w at distance $d' \leq d < \ell_p$ to z in T_z. Thus, if the unique u-w-path in T_z is not alternating or its length is not ℓ_p, then I is not a yes-instance. But otherwise, Tree Rule 2 is applicable to u and w, contradicting irreducibility. □

Lemma 6 *Tree Rule 3 is correct, that is, the instance* $I = (G, \omega, M, \sigma_p, \sigma_c, \ell_p, \ell_c)$ *is a yes-instance if and only if the result of applying Tree Rule 3 to I is.*

Proof Let v and e be as defined in Tree Rule 3 and let $e = \{u, v\}$. To show correctness of Tree Rule 3, we prove that all optimal solutions for I contain e. To this end, let S be an optimal solution with $e \notin S$. By Lemma 3, T_v is an alternating path p ending in v with $|p| < \ell_p$. By definition, $M(u)$ is on p, so $|p| \geq 2$. But then, $G - e$ contains an isolated path of length strictly less than ℓ_p, implying that S is not a solution for I. □

Lemma 7 *Path Rule 1 is correct, that is, the instance* $I = (G, \omega, M, \sigma_p, \sigma_c, \ell_p, \ell_c)$ *is a yes-instance if and only if the result* $I' = (G', \omega, M', \sigma_p - 1, \sigma_c, \ell_p, \ell_c)$ *of applying Path Rule 1 to I is.*

Proof Let $p = (u_0, u_1, \ldots, u_{2l_p+2})$ be as in Path Rule 1 and let $p' := (u_0, u_1, \ldots, u_{\ell_p+1})$. Note that $e_{\ell_p+1}, e_{\ell_p+2}, \ldots$ exist in G'.

"⇒": Let S be a solution for I, let $p' := (u_0, u_1, \ldots, u_{\ell_p+1})$ and note that no cycle of $G[S \cup M]$ contains p since $|p| > \ell_c$. We show that $S' := S \setminus p'$ is a solution of for I' with $\omega'(S') = \omega(S)$. First, note that all vertices of G' are covered by S'. Second, note that $p \setminus p'$ is alternating since $|p| = \ell_p + 1$. Finally, $G'[S' \cup M']$ contains one path less than $G[S \cup M]$.

"⇐": Let S' be a solution for I' and let u denote the vertex onto which $u_0, u_1, \ldots, u_{\ell_p+1}$ have been contracted in I' and let $e := uu_{\ell_p+2}$. We construct a solution S for I with $\omega(S) = \omega'(S')$. First, since $|p| > \ell_c + \ell_p + 1$, no cycle in $G[S \cup M]$ contains e. If $e \notin S' \cup M'$, then $S := S' \cup (p' - e_0)$. If $e \in S' \cup M'$, then $e_j \notin S' \cup M'$ for some $\ell_p < j \leq 2\ell_p$. Then, $S := S' \cup (p' - e_{j-(\ell_p+1)})$ □

Lemma 8 *Path Rule 2 is correct, that is, the instance* $I = (G, \omega, M, \sigma_p, \sigma_c, \ell_p, \ell_c)$ *is a yes-instance if and only if the result* $I' = (G', \omega', M', \sigma_p - 1, \sigma_c, \ell_p, \ell_c)$ *of applying Path Rule 2 to I is.*

Proof First, note that no solution for I or I' covers w in a cycle since T_w is necessarily covered by a path. Also note that q exists in I'.

"⇒": Let S be a solution for I and note that there is some $i \leq \ell_p$ such that $e_i \notin S \cup M$ (in fact, $i \in \{\ell_p, \gamma\}$). Then, however, $f_{i-\gamma} \notin S$ and $S' := S \setminus p$ is a solution for I' and $\omega'(S') = \omega(S)$.

"⇐": Let S' be a solution for I' and note that there is some $i \leq \ell_p - \gamma$ such that $f_i \notin S' \cup M'$ (in fact, $i \in \{0, \ell_p - \gamma\}$). But then, $S' \cup (p - e_{i+\gamma})$ is a solution for I and $\omega(S) = \omega'(S')$. □

Lemma 9 *Path Rule 3 is correct, that is, the instance* $I = (G, \omega, M, \sigma_p, \sigma_c, \ell_p, \ell_c)$ *is a yes-instance if and only if the result* $I' = (G', \omega, M, \sigma_p - 1, \sigma_c, \ell_p, \ell_c)$ *of applying Path Rule 3 to I is.*

Proof "⇒": Let S be solution for I and let q_u and q_v denote the paths containing u and v, respectively, in $G[S \cup M]$.

Case 1: Neither q_u nor q_v contain edges of p. Then, $\gamma = 1$ and $ux_1 \notin S \cup M$. If $\gamma_u + \gamma_v + 1 = \ell_p$, then Case (4) applies and otherwise, Case (5) applies. In both cases, S is also a solution for I'.

Case 2: q_v contains edges of p, but q_u does not. Then, $\gamma_v + \gamma = \ell_p + 1$ and $ux_1 \notin S \cup M$. If $\gamma_u + \gamma = \ell_p + 1$, then Case (3) applies and otherwise, Case (1) applies. In both cases, S is also a solution for I'.

Case 3: q_u contains edges of p, but q_v does not. Then, $\gamma_u + \gamma = \ell_p + 1$ and $vx_{|p|-1} \notin S \cup M$ and $ux_1 \in S \setminus M$. If $\gamma_u + \gamma = \ell_p + 1$, then Case (3) applies and switching ux_1 for wx_1 in S gives a solution of same weight for I'. Otherwise, Case (2) applies and S is also a solution for I'.

Case 4: Both q_u and q_v contain edges of p. Then, $\gamma_u + \gamma_v + \gamma \equiv \ell_p \bmod (\ell_p + 1)$ and $ux_1 \in S \setminus M$. If $\gamma \neq 1$, then Case (6) applies and S is also a solution for I'. Otherwise, Case (3) applies and switching ux_1 for wx_1 in S gives a solution of same weight for I'.

"⇐": Let S' be a solution for I'. Note that, if G' does not contain wx_1, then S is clearly also a solution for I. Thus, assume that G' contains wx_1 and, thus, either Case (3) or (4) applies to I. We show that the result S of switching wx_1 for ux_1 in S' is a solution of the same weight for I. To show this, it suffices to show that, if a path q contains wx_1, then q ends at u. Assume this is false, that is, q contains wx_1 and some edge $e \in E(G') \setminus M$ incident with u. Since T_w is not empty, no cycle in $G'[S' \cup M]$ contains u. Then, the length of the u-v-path containing $p - ux_1$ in G' is equivalent to $\gamma + \gamma_u$ modulo $\ell_p + 1$.

Case 1: $vx_{|p|-1} \in S \cup M$. Then, since q does not end in u, we have $\gamma_u + \gamma + \gamma_v \not\equiv \ell_p \bmod (\ell_p + 1)$. Thus, Case (4) does not apply to I, implying that Case (3) applies to I. But then, $\gamma + \gamma_v = \ell_p + 1$ and, hence, $wx_1 \notin S \cup M$.

Case 2: $vx_{|p|-1} \notin S \cup M$. Then, since q does not end in u, we have $\gamma_u + \gamma \not\equiv 0 \bmod (\ell_p + 1)$, implying $\gamma_u + \gamma \neq \ell_p + 1$ since $0 < \gamma \leq \ell_p$ and $0 < \gamma_u < \ell_p$. Thus, Case (3) does not apply to I, implying that Case (4) applies to I. But then, $\gamma = 1$ and, since $vx_{|p|-1} \notin S \cup M$, we have $wx_1 \notin S \cup M$. □

Observation 1 *Let G be connected. Then, $|V^+| \leq 2\,\mathrm{FES}(G)$ since $|E^+| \leq |V^+| + \mathrm{FES}(G^+) \leq |V^+| + \mathrm{FES}(G)$ and $2|E^+| \geq 3|V^+|$.*

Proof of Theorem 2 By Lemma 4, we have $|V| \leq \ell(|V^+| + 3|E^+|) \overset{\text{Observation 1}}{\leq} \ell(2\,\mathrm{FES}(G) + 3(\mathrm{FES}(G) + 2\,\mathrm{FES}(G)) = 11\ell\,\mathrm{FES}(G)$ and, thus, we obtain $|E| \leq 11\ell\,\mathrm{FES}(G) + \mathrm{FES}(G) = (11\ell + 1) \cdot \mathrm{FES}(G)$. □

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
MW, AC and RG conceived the method and the proofs. MW implemented and tested the algorithms on all datasets. MW, AC and RG wrote the paper.

Acknowledgements
Publication of this work is funded by the Institut de Biologie Computationnelle.
This article has been published as part of *BMC Bioinformatics* Volume 16 Supplement 14, 2015: Proceedings of the 13th Annual Research in Computational Molecular Biology (RECOMB) Satellite Workshop on Comparative Genomics: Bioinformatics. The full contents of the supplement are available online at http://www.biomedcentral.com/bmcbioinformatics/supplements/16/S14.

Authors' details
[1]Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM) - Université de Montpellier - UMR 5506 CNRS, 161 rue Ada, 34090 Montpellier, France. [2]Institut de Biologie Computationnelle, Lirmm Bât 5 - 860 rue de St Priest, 34090 Montpellier, France.

Published: 2 October 2015

References
1. Reddy TBK, Thomas AD, Stamatis D, Bertsch J, Isbandi M, Jansson J, Mallajosyula J, Pagani I, Lobos EA, Kyrpides NC: **The Genomes OnLine Database (GOLD) v.5: a metadata management system based on a four level (meta)genome project classification.** *Nucleic Acids Research* 2014, **43(D1)**:1099-1106[https://gold.jgi-psf.org/distribution].
2. Donmez N, Brudno ML: **SCARPA: scaffolding reads with practical algorithms.** *Bioinformatics* 2013, **29(4)**:428-434.
3. Gritsenko AA, Nijkamp JF, Reinders MJT, de Ridder D: **GRASS: a generic algorithm for scaffolding next-generation sequencing assemblies.** *Bioinformatics* 2012, **28(11)**:1429-1437.
4. Dayarian A, Michael TP, Sengupta AM: **SOPRA: Scaffolding algorithm for paired reads via statistical optimization.** *BMC Bioinformatics* 2010, **11**:345.
5. Gao S, Sung W-K, Nagarajan N: **Opera: Reconstructing Optimal Genomic Scaffolds with High-Throughput Paired-End Sequences.** *Journal of Computational Biology* 2011, **18(11)**:1681-1691.
6. Salmela L, Mäkinen V, Välimäki N, Ylinen J, Ukkonen E: **Fast scaffolding with small independent mixed integer programs.** *Bioinformatics* 2011, **27(23)**:3259-3265.
7. Boetzer M, Henkel CV, Jansen HJ, Butler D, Pirovano W: **Scaffolding pre-assembled contigs using SSPACE.** *Bioinformatics* 2011, **27(4)**:578-579.
8. Koren S, Treangen TJ, Pop M: **Bambus 2: Scaffolding metagenomes.** *Bioinformatics* 2011, **27(21)**:2964-2971.
9. Hunt M, Newbold C, Berriman M, Otto T: **A comprehensive evaluation of assembly scaffolding tools.** *Genome Biology* 2014, **15(3)**.
10. Huson DH, Reinert K, Myers EW: **The greedy path-merging algorithm for contig scaffolding.** *Journal of the ACM* 2002, **49(5)**:603-615.
11. Solinhac R, Leroux S, Galkina S, Chazara O, Feve K, Vignoles F, Morisson M, Derjusheva S, Bed'hom B, Vignal A, Fillon V, Pitel F: **Integrative mapping analysis of chicken microchromosome 16 organization.** *BMC Genomics* 2010, **11(1)**:616.
12. Sharpton TJ: **An introduction to the analysis of shotgun metagenomic data.** *Frontiers in Plant Science* 2014, **5(209)**.
13. Shmoys DB, Lenstra JK, Kan AHGR, Lawler EL: **The Traveling Salesman Problem: a Guided Tour of Combinatorial Optimization.**John Wiley & Sons 1985.
14. Tutte WT: **A short proof of the factor theorem for finite graphs.** *Canadian Journal of Mathematics* 1954, **6**:347-352.
15. Chateau A, Giroudeau R: **Complexity and Polynomial-Time Approximation Algorithms around the Scaffolding Problem.** In *AlCoB Lecture Notes in Computer Science. Volume 8542.* Springer;Dediu, A.H., Martín-Vide, C., Truthe, B. 2014:47-58.
16. Chateau A, Giroudeau R: **A complexity and approximation framework for the maximization scaffolding problem.** *Theoretical Computer Science* 2015, **595**:92-106, doi:10.1016/j.tcs.2015.06.023.
17. Weller M, Chateau A, Giroudeau R: **On the implementation of polynomial-time approximation algorithms for scaffold problems.** submitted 2015.
18. Nieuwerburgh FV, Thompson RC, Ledesma J, Deforce D, Gaasterland T, Ordoukhanian P, Head SR: **Illumina mate-paired DNA sequencing-library preparation using Cre-Lox recombination.** *Nucleic Acids Research* 2012, **40(3)**:24-24.
19. Boetzer M, Henkel CV, Jansen HJ, Butler D, Pirovano W: **Scaffolding pre-assembled contigs using SSPACE.** *Bioinformatics* 2011, **27(4)**:578-579.

20. Reed B, Smith K, Vetta A: **Finding odd cycle transversals.** *Operations Research Letters* 2004, **32(4)**:299-301.
21. Sahlin K, Vezzi F, Nystedt B, Lundeberg J, Arvestad L: **BESST - efficient scaffolding of large fragmented assemblies.** *BMC Bioinformatics* 2014, **15(1)**:281.
22. Luhmann N, Chauve C, Stoye J, Wittler R: **Scaffolding of ancient contigs and ancestral reconstruction in a phylogenetic framework.** *Proceedings of Brazilian Symposium on Bioinformatics Lecture Notes in Computer Science* 2014, **8826**:135-143.
23. Rajaraman A, Tannier E, Chauve C: **FPSAC: Fast Phylogenetic Scaffolding of Ancient Contigs.** *Bioinformatics* 2013, **29(23)**:2987-2994.
24. Aganezov S, Sitdykovaa N, Alekseyev MA: **AGCConsortium: Scaffold assembly based on genome rearrangement analysis.** *Computational Biology and Chemistry* 2015, **57**:46-53.
25. Impagliazzo R, Paturi R, Zane F: **Which Problems Have Strongly Exponential Complexity?** *Journal of Computer and System Sciences* 2001, **63(4)**:512-530.
26. Bodlaender HL, Cygan M, Kratsch S, Nederlof J, Impagliazzo R, Paturi R, Zane F: **Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth.** *Information and Computation* 2015, **243**:86-111.
27. Downey RG, Fellows MR: **Fundamentals of Parameterized Complexity.** *Texts in Computer Science* Springer; 2013.
28. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R: **The sequence alignment/map format and SAMtools.** *Bioinformatics* 2009, **25(16)**:2078-2079.
29. Chikhi R, Rizk G: **Space-efficient and exact de bruijn graph representation based on a bloom filter.** *Algorithms for Molecular Biology* 2013, **8**:22.
30. Li H, Durbin R: **Fast and accurate long-read alignment with Burrows-Wheeler transform.** *Bioinformatics* 2010, **26(5)**:589-595.
31. Briot N, Chateau A, Coletta R, Givry SD, Leleux P, Schiex T: **An Integer Linear Programming Approach for Genome Scaffolding.** *Workshop Constraints in Bioinformatics* 2014.
32. Salzberg SL, Phillippy AM, Zimin A, Puiu D, Magoc T, Koren S, Treangen TJ, Schatz MC, Delcher AL, Roberts M, Marc¸ais G, Pop M, Yorke JA: **GAGE: A critical evaluation of genome assemblies and assembly algorithms.** *Genome Research* 2012, **22(3)**:557-567[http://gage.cbcb.umd.edu].
33. Zerbino DR, Birney E: **Velvet: algorithms for de novo short read assembly using de Bruijn graphs.** *Genome research* 2008, **18(5)**:821-829.
34. Langmead B, Trapnell C, Pop M, Salzberg SL: **Ultrafast and memory-efficient alignment of short DNA sequences to the human genome.** *Genome Biology* 2009, **10(3)**:25.
35. Chikhi R, Rizk G: **Space-efficient and exact de Bruijn graph representation based on a Bloom filter.** *Algorithms for Molecular Biology* 2013, **8(22)**.
36. 2013 [http://bioinf.dimi.uniud.it/variathon], Variathon.
37. Denton JF, Lugo-Martinez J, Tucker AE, Schrider DR, Warren WC, Hahn MW: **Extensive error in the number of genes inferred from draft genome assemblies.** *PLoS Computational Biology* 2014, **10(2)**:1003998.
38. Koster A: **TOL - Treewidth Optimization Library.** 2012.
39. Morgulis A, Coulouris G, Raytselis Y, Madden TL, Agarwala R, Scha¨ ffer AA, Koster A: **Database indexing for production MegaBLAST searches.** *Bioinformatics (Oxford, England)* 2008, **24(16)**:1757-1764.
40. Kolodner R, Tewari KK: **Inverted repeats in chloroplast DNA from higher plants*.** *Proceedings of the National Academy of Sciences of the United States of America* 1979, **76(1)**:41-45.

doi:10.1186/1471-2105-16-S14-S2
Cite this article as: Weller *et al.*: Exact approaches for scaffolding. *BMC Bioinformatics* 2015 **16**(Suppl 14):S2.