Linearizing Genomes: Exact Methods and Local Search

Tom Davot, Annie Chateau, Rodolphe Giroudeau, Mathias Weller

Linearizing Genomes: Exact Methods and Local Search

Tom Davot[1], Annie Chateau[1], Rodolphe Giroudeau[1], Mathias Weller[2]

[1] LIRMM - CNRS UMR 5506 - Montpellier, France
[2] CNRS, LIGM (UMR 8049), Champs-s/-Marne, France
{davot,chateau,rgirou}@lirmm.fr, mathias.weller@u-pem.fr

Abstract. In this article, we address the problem of genome linearization from the perspective of Polynomial Local Search, a complexity class related to finding local optima. We prove that the linearization problem, with a neighborhood structure, the neighbor slide, is PLS-complete. On the positive side, we develop two exact methods, one using tree decompositions with an efficient dynamic programming, the other using an integer linear programming. Finally, we compare them on real instances.

1 Introduction

Motivation. When inferring genome sequences from high-throughput sequencing (HTS) data, we obtain (after assembly) fragments of the target sequence called *contigs*[3] without any information on how these contigs are located in the genome. To address this shortcoming, contigs can be linked using external information (usually a read-pairing included in the HTS data), yielding a graph (called *scaffold graph*) whose vertices are contig extremities and edges are either contigs or links between them. The *scaffolding* operation then aims at selecting the best paths in this graph in order to produce longer genomic sequences called scaffolds. Previous work focuses on the production of sequences by solving the so-called SCAFFOLDING problem in this graph [4, 14, 16]. Scaffolding is a widely studied problem in bioinformatics and can be modeled by numerous, mostly heuristic, methods [8].

Unfortunately, real-world genomes escape the relative simplicity of previous models (that still lead to NP-complete problems). A particular problem is modeling contigs occurring multiple times in the target genome. Such “repeats” and their “multiplicity” (or “copy numbers”) vary depending on the species and individual [2]. Due to the conservatism of some assembly methods, a repeat may cover an entire contig which is separated from the other genomic side fragments [11]. Recent methods address this problem, avoiding chimeric reconstruction by using long reads as additional data [3, 13]. Unfortunately, most projects

Preprint version.
The final authenticated version is available online at https://doi.org/10.1007/978-3-030-38919-2_41.

[3] Contigs are words on a genomic alphabet, usually $\{A, C, G, T\}$.

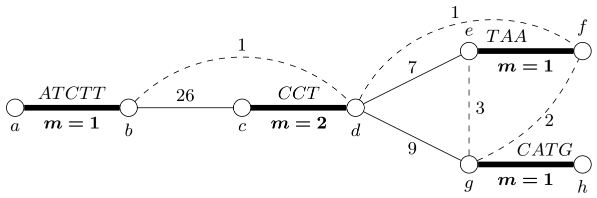

Fig. 1. A scaffold graph and a solution graph obtained with an optimal solution of SCAM. Matching edges are bold and plain edges are part of the solution graph. Edge cd has multiplicity two. Other contigs have multiplicity one. Edges of the solution graph also have multiplicity one. Links between contigs are labeled by their weight. Because of the presence of the ambiguous path cd, two optimal solutions are possible for SCAM with $\sigma_c = 0$ and $\sigma_p = 2$: $\{(a, b, c, d, e, f), (c, d, g, h)\}$ and $\{(a, b, c, d, g, h), (c, d, e, f)\}$.

on genomic databases are still constituted of short-reads only and are not intended to be resequenced with long-reads technologies in the near future. One motivation of our work it to take care of these kind of projects, and improve assemblies using only the original short-read (though paired-end) data. In this context, a solution to the Scaffolding problem may not be a collection of distinct paths, but rather a graph, called *solution graph* (which is a particular scaffold graph with multiplicities). Transforming such a graph into genomic sequences turns out to be a challenging task. The aim of the present work is to study the problem consisting of removing ambiguities in the solution graph in order to provide longer and error-free genomic sequences with minimal loss of information.

In the following, most of the proofs has been omitted due to space constraints. A full version including the proofs is available in https://hal-lirmm.ccsd.cnrs.fr/lirmm-02332049.

2 Notation and Problem Description

With G denoting a graph, we let $V(G)$ and $E(G)$ be the sets of vertices and edges of G, respectively. A scaffold graph (G, M^*, ω, m') is an edge-weighted, simple, undirected graph G equipped with 1. a perfect matching M^* that corresponds to the contigs, 2. non-contig edges uv whose weights $\omega(uv)$ indicate the likelihood that the contig-extremity u is adjacent to the contig-extremity v in the target genome and 3. a multiplicity m' on contig edges which indicates the desired number of their occurrences (see Figure 1). An alternating walk $(u_0, \ldots, u_{2\ell-1})$ is a sequence of vertices such that for each $i < \ell$, $u_{2i}u_{2i+1} \in M^*$ and $u_{2i+1}u_{2i+2} \in E(G) \setminus M^*$. If $u_0 = u_{2\ell-1}$, the walk is *closed*. The Scaffolding with Multiplicities problem is stated as follows:

Scaffolding with Multiplicities (SCAM)
Input: a scaffold graph (G, M^*, ω, m') and $\sigma_\mathrm{p}, \sigma_\mathrm{c}, k \in \mathbb{N}$
Question: Is there a multiset S of at most σ_c closed and at most σ_p non-closed alternating walks in G such that each $e \in M^*$ occurs $m'(e)$ times across all walks of S and $\sum_{e \in E(S) \setminus M^*} \omega(e) \geq k$?

In this work, we will not focus on the SCAM problem itself (instead, the reader is referred to Weller et al. [14, 16, 17]). Instead, we assume that we are given a solution S, whose walks then induce a subgraph of the input scaffold graph which we call *solution graph* (G^*, M^*, ω, m). More precisely, given a scaffold graph (G, M^*, ω, m) and a solution S of SCAM, the solution graph (G^*, M^*, ω, m) is obtained by removing the edges that do not belong to S. The multiplicity function m defined on all the edges of the solution graph is the number of times that an edge occurs in S. Note that for each matching edge e, we have $m(e) = m'(e)$. It turns out that, in presence of repeated contigs, a solution graph implies a unique set of sequences if and only if it does not contain so called *ambiguous paths* [15] (see Figure 1 for an example).

Definition 1 (Ambiguous path). *Let p be path with extremities u and v in a solution graph. If, for all vertices x of p, p also contains the matching edge containing x, we call p* alternating. *If all edges of p have the same multiplicity μ (that is, $m(e) = \mu$ for all $e \in p$), then p is called μ-*uniform *(or simply* uniform *is μ is unknown). If p is alternating, uniform, and both of u and v are incident with a non-matching edge of multiplicity strictly less than μ, then p is called* ambiguous.

To break ambiguous paths, we remove non-contig edges from the solution graph, thereby losing information, and our goal is to minimize this loss. Definition 1 implies that minimal solutions remove all incident non-contig edges from a selected set X of vertices. The "cost" of such a set X can be defined by the following scorings:

Cut score. Pay one per vertex in X: $\text{score}(X) := |X|$.
Path score. Pay one for each multiplicity that is removed:
$\text{score}(X) := \sum\{m(uv) \mid uv \in E \setminus M^* \wedge uv \cap X \neq \varnothing\}$.
Weight score. Pay the total cost of edges that are removed:
$\text{score}(X) := \sum\{m(uv) \cdot \omega(uv) \mid uv \in E \setminus M^* \wedge uv \cap X \neq \varnothing\}$.

Since the Path score and the Weight score are very similar, we study in this paper only the Cut score and the Weight score. The following reduction rules simplify a given instance (solution graph) without changing the solution set X.

Rule 1 ([15]) *Let p be a μ-uniform alternating path with extremities u and v. Remove p and add a new contig edge uv with multiplicity μ.*

Rule 2 ([15]) *Let $uv \in M^*$ be a contig edge not appearing in ambiguous paths and let u and v have degree at least two. Then, remove uv, add new vertices u' and v' and add the contig edges uv' and vu' with multiplicity $m(uv)$.*

Let (G^*, M^*, ω, m) be a solution graph and let $u \in V(G^*)$. We let $N_{G^*}(u) = \{v \mid uv \in E(G^*) \setminus M^*\}$ denote the set of neighbors of u linked to u with a non-matching edge. We say that a vertex u is *clean* if $N_{G^*}(u) = \varnothing$ and a matching edge $uv \in M^*$ is *clean* if at least one of its extremities is clean. In the following, we assume that all solution graphs are reduced with respect to Rule 1, and we observe that, in this case, all ambiguous paths have length one. Thus, we use the term “ambiguous edges” (resp. “non-ambiguous edges”) when we speak of ambiguous (resp. non-ambiguous) paths. With Rule 2, we can further assume that all non-ambiguous edges are clean, implying that each matching edge e is ambiguous if and only if e is not clean. Hence, disambiguating a solution means to render all matching edges clean. We can now formulate our problem Semi-Brutal Cut as follows.

Semi-Brutal Cut (SBC)
Input: A solution graph (G^*, M^*, ω, m) and some $k \in \mathbb{N}$
Question: Is there a set X of extremities of ambiguous edges in G^* such that removing all non-matching edges incident to vertices of X renders all matching edges clean, and $\text{score}(X) \leq k$?

For a vertex u of G^*, we let $\omega(u)$ denote the sum of the weights of all non-matching edges incident to u. For a solution X of Semi-Brutal Cut, we let $\omega(X) := \sum_{u \in X} \omega(u)$. We say that u is *cut* if $u \in X$. Since we are not limited in number of cuts for the weight score, we suppose that in a solution X for Semi-Brutal Cut under the weight score, each ambiguous edge of (G^*, M^*, ω, m) contains exactly one vertex in X.

3 Related Work

Problems similar to the linearization of scaffolds are studied in the context of guided, multiple-source assembly problems [12]. However, the model does not integrate multiplicities as a constraint on the structure of the desired paths. In previous work, we show that the variants of Semi-Brutal Cut according to all presented scoring functions are NP-complete [15]. In [10], we explore special classes of graphs, namely bipartite, planar with bounded degree, analyzing complexity and approximability, showing that even in very restricted cases, the problem is hard to solve. We also proposed a 2-approximation algorithm under the weight score and a 4-approximation under the cut score. In the present work, we consider general instances, showing that even finding a locally optimal solution is hard, but propose effective exact methods to linearize genomes.

4 Hardness using PLS-reduction

This section is devoted to determine the local-search complexity using the PLS (Polynomial Local Search) class, which models the difficulty of finding a locally optimal solution to an optimization problem [5]. Schäffer and Yannakakis [9] proved several classic combinatorial optimization problems PLS-complete. In the following, we propose a new neighborhood structure called the *neighbor slide* adapted to Semi-Brutal Cut. We recall first some definitions related to PLS. A *neighborhood structure* N is a function that associates to each solution S a set of solutions $N(S)$. A *local search problem* is a combinatorial optimization problem P for which, given a neighborhood structure N, we want to find a solution S (called *local optimum*), such that no solution in $N(S)$ has a better score. In the following, we let P/N denote a local search problem where P is a combinatorial optimization problem and N a neighborhood structure.

Definition 2 (PLS). *A local search problem P/N is in* PLS *if there are polynomial-time algorithms A_L, B_L, and C_L such that*
(a) for each instance x, A_L gives an initial solution S_{init},
(b) for each solution S, B_L determines the score of S, and
(c) for each solution S, C_L determines if S is a local optimum and, if not, gives a solution with the best score in $N(S)$.

To show that finding a local optimum for a problem P_1/N_1 is at least as difficult as finding a local optimum for a problem P_2/N_2, we use *PLS-reductions.*

Definition 3 (PLS-reduction). *A local search problem P_1/N_1 is PLS-reducible to a local search problem P_2/N_2 if there are polynomial-time computable functions f and g such that:*
(a) If x_1 is an instance of P_1, then $f(x_1)$ is an instance of P_2.
(b) If S_2 is a solution for $f(x_1)$, then $g(x_1, S_2)$ is a solution for x_1.
(c) If S_2 is a local optimum for $f(x_1)$, then $g(x_1, S_2)$ is a local optimum for x_1.

Then, a local search problem P/N is *PLS-complete* if P/N is in *PLS* and every problem in *PLS* can be *PLS*-reduced to P/N. We now introduce the Max W2SAT problem and the *Flip* neighborhood structure.

Max W2SAT
Input: A boolean formula φ in conjunctive normal form where each clause C_i has a weight $\omega(C_i)$ and contains exactly two variables.
Task: Find an assignment maximizing the total weight of satisfied clauses.

Definition 4 (Flip). *Let S be a solution for φ. A solution S' is in $N(S)$ if there exists a unique variable x_i such that the assignment of x_i is different in S and S'. We say that S' is obtained by flipping the value of x_i in S.*

Note that Max W2SAT/Flip is PLS-complete [7]. Let φ be an instance of Max W2SAT and let S be a solution for φ. We let $\omega(\varphi)$ denote the sum of the weights of all clauses of φ and we let $\omega(S)$ denote the total weight of all clauses that are *not* satisfied by S. From an instance of Max W2SAT, we build an instance of SBC using the following construction.

Construction 1 (See Figure 2(left)) *Let φ be an instance of* Max W2SAT *with n' variables x_i and m' clauses C_j and let $occ(x_i)$ denote the number of occurrences of x_i in φ. We construct the following solution graph (G^*, M^*, ω, m).*
1. Construct a matching edge s_1s_2 with $m(s_1s_2) = 2m'$.
*2. For each x_i, construct a matching edge $u_i\overline{u}_i$ such that $m(u_i\overline{u}_i) = occ(x_i) + 1$ (*variable edge*).*
*3. For each clause C_j, construct a matching edge $v_j^1v_j^2$ such that $m(v_j^1v_j^2) = 2$ (*clause edge*).*
4. For each clause C_j, let x_k be the t^{th} variable of the clause. If x_k occurs positively in the clause, then add the edge $v_j^tu_k$ with $m(v_j^tu_k) = 1$ and $\omega(v_j^tu_k) = \omega(C_j)$. Otherwise, add the edge $v_j^t\overline{u}_k$ with $m(v_j^t\overline{u}_k) = 1$ and $\omega(v_j^t\overline{u}_k) = \omega(C_j)$.

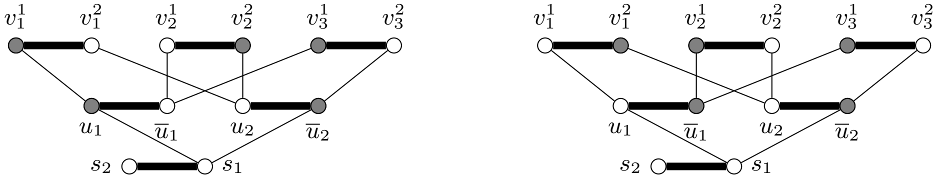

Fig. 2. Left: The graph produced by Construction 1 on input $\varphi = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ (each clause has weight one). Matching edges are bold and all non-matching edges have weight one. A solution S with $\omega(S) = 6$ is highlighted in gray. In S, $v_1^1 v_1^2$ is satisfied, $v_2^1 v_2^2$ is unsatisfied and $v_3^1 v_3^2$ is neither satisfied nor unsatisfied. **Right:** A solution of weight 5 produced by a neighbor slide of $u_1\overline{u}_1$.

5. *Finally, for each matching edge $u_i\overline{u}_i$, if $\omega(u_i) < \omega(\overline{u}_i)$, add an edge s_1u_1 with $m(s_1u_1) = 1$ and $\omega(s_1u_1) = \omega(\overline{u}_i) - \omega(u_i)$. If $\omega(u_i) > \omega(\overline{u}_i)$, add an edge $s_1\overline{u}_1$ with $m(s_1\overline{u}_1) = 1$ and $\omega(s_1\overline{u}_1) = \omega(u_i) - \omega(\overline{u}_i)$.*

Note that for each variable edge $u_i\overline{u}_i$, we have $\omega(u_i) = \omega(\overline{u}_i)$. All matching edges except s_1s_2 are ambiguous. If a cut in a clause edge $v_j^1v_j^2$ is adjacent to a cut in a variable edge, then we say that the clause edge $v_j^1v_j^2$ is *satisfied*. If no extremity of a clause edge $v_j^1v_j^2$ is adjacent to a cut vertex in a variable edge, we say that the clause edge $v_j^1v_j^2$ is *unsatisfied*. Note that a clause edge could be neither satisfied nor unsatisfied. In a graph produced by Construction 1, we simulate the flipping operation with the neighbor slide operation defined as follows:

Definition 5 (Neighbor Slide, see Figure 2). *Let $S \subseteq V(G^*)$ be a solution for (G^*, M^*, ω, m) and let uv be an unclean matching edge of G^* with $u \in S$. The* neighbor slide *operation applied to uv produces a new solution S' as follows:*
1. $S' \leftarrow (S \cup \{v\}) \setminus \{u\}$,
2. for each neighbor $n_u \neq s_1$ of u: $S' \leftarrow (S' \cup \{M^*(n_u)\}) \setminus \{n_u\}$, and
3. for each neighbor $n_v \neq s_1$ of v: $S' \leftarrow (S' \cup \{n_v\}) \setminus \{M^*(n_v)\}$.
Thus, a solution S' belongs to $N(S)$ if S' can be produced by applying a neighbor slide operation on S.

Definition 6. *Let φ be an instance of* Max W2SAT*, let (G^*, M^*, ω, m) be the graph produced by Construction 1, and let X be a solution for it. A solution S for φ* corresponds *to X if, for all matching edges $u_i\overline{u}_i$, we have $u_i \in X \Rightarrow S(x_i) = 1$ and $\overline{u}_i \in X \Rightarrow S(x_i) = 0$.*

Note that, after a neighbor slide of a variable edge, all adjacent clause edges are either satisfied or unsatisfied, and that if a clause edge $v_j^1v_j^2$ is satisfied (resp. unsatisfied), then the corresponding clause C_j is satisfied (resp. unsatisfied).

Lemma 1. *Let X be a solution for (G^*, M^*, ω, m), produced by Construction 1 and let S be the corresponding solution for φ.*
1. *If X is a local minimum, then all clause edges are satisfied or unsatisfied.*
2. *If all clause edges are satisfied or unsatisfied by X, then $\omega(X) = \omega(\varphi) + \omega(S)$.*

Proof.

1. Suppose there is a clause edge $v_j^1 v_j^2$ that is neither satisfied nor unsatisfied. Thus, it exists a cut vertex adjacent to $v_j^1 v_j^2$ that is not adjacent to the cut vertex of $v_j^1 v_j^2$. By neighbor-slidding the clause edge $v_j^1 v_j^2$ we can produce a solution with a smaller weight, contradicting the fact that X is a local minimum.
2. Let $u_i \overline{u}_i$ be a variable edge and ψ_i be the list of the clauses where the variable x_i occurs. We have $\omega(u_i) = \omega(\overline{u}_i) = \sum_{C_j \in \psi_i} \omega(C_j)$.
   Thus, the sum of the weights removed by the cuts in the variable edges is equal to $\sum_{i \leq n'} \sum_{C_j \in \psi_i} \omega(C_j) = \omega(\varphi)$. Let $v_j^1 v_j^2$ be a clause edge. If $v_j^1 v_j^2$ is satisfied, then its cut does not increase the weight of X since the non-matching edge incident to this cut is already removed by the cut in the variable edge. If $v_j^1 v_j^2$ is unsatisfied, then it cut increases the weight of X by the weight of C_j. Since the sum of weights removed by the cuts in the unsatisfied clauses edges correspond to the weight of S, we have $\omega(X) = \omega(\varphi) + \omega(S)$.

Theorem 1. *SBC/Neighbor slide is PLS-complete for the weight score.*

Proof. It is easy to see that SBC/Neighbor slide is in PLS. We propose a PLS-reduction of Max W2SAT/Flip to SBC/Neighbor slide. Let φ be an instance of Max W2SAT. The function defined by Construction 1 produces an instance (G^*, M^*, ω, m) of Semi-Brutal Cut and the function defined in Definition 6 computes a solution for φ from a solution for (G^*, M^*, ω, m). It remains to show that, if a solution X is a local minimum of Semi-Brutal Cut in (G^*, M^*, ω, m), then its corresponding solution S is also a local minimum. By Lemma 1(1) and Lemma 1(2), we have $\omega(X) = \omega(\varphi) + \omega(S)$. Suppose that S is not a local minimum. Then, there is a variable x_i in S such that flipping its value produces a solution S'_1 with a smaller weight. Let S'_2 be the solution produced by the neighbor-slide operation on the variable edge $u_i \overline{u}_1$ in X. Note that the corresponding solution of S'_2 is S'_1. By Lemma 1(1), all clause edges in X are either satisfied or unsatisfied and since the clause edges modified by a neighbor-slide are either satisfied or unsatisfied, all clause edges in S'_2 are either satisfied or unsatisfied. Thus, by Lemma 1(2), $\omega(S'_2) = \omega(\varphi) + \omega(S'_1) < \omega(X)$, contradicting the fact that X is a local minimum.

5 Exact Methods

5.1 Integer Linear Programming

In this section, we propose an integer linear program modeling Semi-Brutal Cut for all scores.

Variables. For each non-matching edge e_k, we define a binary variable x_k which equals 1 if and only if one of its extremities is in the solution, that is, e_k is removed from the graph. For each extremity u_i of an ambiguous edge p, we define two binary variables c_i and n_i. $c_i = 1$ iff u_i is in the solution and $n_i = 1$ if and only if all neighbors $v \neq M^*(u_i)$ of u_i are in the solution.

Constraints.

(1) For any ambiguous matching edge $u_i u_j$, we force one of the extremities to have degree one by adding the constraint $n_i + n_j + c_i + c_j \geq 1$.

(2) If any extremity u_i is adjacent to a non-ambiguous matching edge, then not all neighbors of u_i can be cut. In this case, we add the constraint $n_\ell = 0$.

(3) For all extremities u_i, we force all neighbors of u_i (except $M^*(u_i)$) to be cut if $n_i = 1$ by adding the constraint $\sum\limits_{u_\ell \in N(u_i)} c_\ell \geq n_i \cdot |N(u_i)|$.

(4) For each extremity u_i of a non-matching edge e_k, we force that e_k is removed from the graph if $u_i = 1$ by adding the constraint $x_k \geq c_i$.

Objective function. For the cut score, we want to minimize the number of vertices in the solution, that is, the number of variable c_i with value one. Thus, the objective function for the cut score is $\min \sum_i c_i$. For the weight score, we want to minimize the total weight of the edges removed from the graph. Thus, the objective function for the weight score is $\min \sum_{e_k \in E(G) \setminus M^*} x_k \cdot \omega(e_k)$.

5.2 Dynamic Programming on Tree Decompositions

We show that Semi-Brutal Cut can be solved in linear time on classes of graphs that exhibit a constant bound on the treewidth, such as series-parallel or outerplanar graphs. To this end, we present a dynamic programming algorithm, working on nice tree decompositions, that finds an optimal solution in $O(2^{tw} \cdot |E(G)|)$ under the weight score and in $O(5^{tw} \cdot |E(G)|)$ under the cut score, where tw is the treewidth of the input graph.

Definition 7 ([6]). *Given a graph G, a* tree decomposition *for G is a pair $(T, \mathcal{X})$ where T is a tree and $\mathcal{X} = \{B_i \mid i \in V(T)\}$ is a multiset of subsets of $V(G)$ (called "bags") such that*
(a) for each $uv \in E(G)$, there is some i with $uv \subseteq B_i$ and
(b) for each $v \in V(G)$, the bags B_i containing v form a connected subset of T.
The width of $(T, \mathcal{X})$ is $\max_i |B_i| - 1$. Further, $(T, \mathcal{X})$ is called nice *if*
(c) T is rooted at bag B_r, with $B_r = \varnothing$ and each bag has at most two children.
(d) Each bag B_i of T has one of the four types:

- ***Leaf bag:*** *i has no children and $B_i = \varnothing$.*
- ***Join bag:*** *i has two children j and k and $B_i = B_j = B_k$.*
- ***Introduce** u **bag:*** *i has only one child j and $B_j = B_i \setminus \{u\}$.*
- ***Forget** u **bag:*** *i has only one child j and $B_j = B_i \cup \{u\}$.*

An example of nice tree decomposition is depicted in Figure 5. For any bag B_i of T, we let G_i denote the subgraph of G induced by the vertices of G that are introduced "below" B_i (that is, in a bag of the subtree of T that is rooted at i).

Note that for each vertex u of G, $(T, \mathcal{X})$ contains exactly one forget u bag. Further, the root r of a nice tree decomposition is a forget bag and we let r' denote the vertex forgotten by r.

Tree Decompositions Introducing Matching Edges. Let (G^*, M^*, ω, m) be a solution graph and let $B_i \in \mathcal{X}$. In our algorithm, we need $M^*(u) \in B_i$ for each

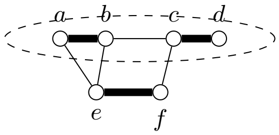

Fig. 3. A subgraph G_i^* with $B_i = \{a, b, c, d\}$ (matching edges in bold). Let $Y_1 = \{(a, ”\varnothing”), (b, ”\varnothing”), (c, ”\times”), (d, ”\times”)\}$, let $Y_2 = \{(a, ”\varnothing”), (b, ”N”), (c, ”\varnothing”), (d, ”\varnothing”)\}$, and let $Y_3 = \{(a, ”N”), (b, ”\times”), (c, ”\varnothing”), (d, ”\varnothing”)\}$. No set vertex set X is eligible for (Y_1, B_i) and (Y_2, B_i) but $\{b, e\}$ is eligible for (Y_3, B_i). The trace of Y_3 is $\mathcal{T}(Y_3) = \{b\}$.

$u \in B_i$. For this reason, we contract all matching edges in (G^*, M^*, ω, m), yielding a graph G' with $V(G') = M^*$. We compute a nice tree decomposition $(T, \mathcal{X})$ of G', then the vertices of G' are expanded, that is, we replace the vertices of G' in the tree decomposition by their corresponding matching edges (see Figure 4). Each introduce u bag now introduces the matching edge $uM^*(u)$. We call such a tree decomposition for G^* *M^*-preserving*. In the following, G' refers to the graph with contracted matching edges and G^* refers to the original solution graph.

Signatures. To every (X, V') where X is a solution of a subgraph H and V' is a subset of vertices of H, we associate a signature describing how the vertices of V' are cut in X. The signature of a vertex u can be ”×”, ”N”, or ”∅”, depending on whether, respectively, u is cut, all neighbors of u are cut, or u is not cut.

Definition 8 (see Figure 3). *Let H be a subgraph of G^* such that, for each $u \in V(H)$, we have $M^*(u) \in H$. Let $X \subseteq V(H)$ be a solution for (G^*, M^*, ω, m) in H and let $V' \subseteq V(H)$. A mapping $Y : V' \to \{”N”, ”\times”, ”\varnothing”\}$ with*
(i) $Y(u) = ”\times” \Leftrightarrow u \in X$ and,
(ii) $Y(u) = ”N” \Rightarrow N_H(u) \subseteq X$
is called signature *of X in V' and $\mathcal{T}(Y) = \{u \mid Y(u) = ”\times”\}$ is called* trace *of Y.*

Note that a solution X can be associated to many signatures. Likewise, two different solutions X and X' of H such that $X \cap V' = X' \cap V'$ are associated to the same signatures. In order to minimize the number of signatures, we add some restrictions on the mappings. The main idea is that sub-solutions with the same signature are equivalently suited to construct a complete solution. Thus, for a vertex set V', we define a set of signatures $\mathcal{Y}(V')$ as follows.

Definition 9. *Let V' be a vertex set. We define $\mathcal{Y}(V')$ as the set of all $Y : V' \to \{”\varnothing”, ”N”, ”\times”\}$ such that, for all $u \in V'$, the three following conditions hold:*

1. *$uM^*(u)$ is clean $\Leftrightarrow Y(u) = Y(M^*(u)) = ”\varnothing”$*
   (no cut occurs in an already clean matching edge).
2. *if the considered scoring function is the weight score, then:*
   - *$Y(u) \neq ”N”$ and,*
   - *$Y(u) = ”\varnothing” \Leftrightarrow Y(M^*(u)) = ”\times”$*
     (each ambiguous edge contains exactly one cut).
3. *if the considered scoring function is the cut score, then:*
   - *$Y(u) = ”\varnothing” \Rightarrow Y(M^*(u)) \neq ”\varnothing”$*
     (an ambiguous edge must be clean) and,
   - *$Y(u) = ”N” \Rightarrow Y(M^*(u)) = ”\varnothing”$*
     (no need to store a neighbor cut if $M^*(u)$ is cut or has a neighbor cut).

Note that if V' contains a single ambiguous edge, then $|\mathcal{Y}(V')| = 2$ under the weight score and $|\mathcal{Y}(V')| = 5$ under the cut score.

Definition 10. *Let $Y_i : V_i \to \{"\varnothing","N","\times"\}$ for $i \in \{1,2\}$ be two signatures such that $V_1 \cap V_2 = \varnothing$. The* union *of Y_1 and Y_2 is the mapping $Y_1 \cup Y_2 : V_1 \cup V_2 \to \{"\varnothing","N","\times"\}$ with*

$$(Y_1 \cup Y_2)(v) = \begin{cases} Y_1(v) & \text{if } v \in V_1 \\ Y_2(v) & \text{otherwise.} \end{cases}$$

For each bag B_i of a given, M^*-preserving tree decomposition of G^*, we will compute solutions for G_i^*. To this end, we introduce the following definition.

Definition 11 (see Figure 3). *Let $(\mathcal{X}, T)$ be a nice tree decomposition of G^*, let $X \subseteq V(G_i^*)$, let $B_i \in \mathcal{X}$, let $Y \in \mathcal{Y}(B_i)$, and let $u \in B_i$. Further, let*

(i) Y be the signature of X in B_i and,
(ii) X be a solution for (G_i^*, M^*, ω, m).

Then, we call X eligible *with respect to (Y, B_i).*

If there is no set eligible for a pair (Y, B_i), we say that the signature Y is *incompatible* with G_i^*.

Lemma 2. *Let $B_i \in \mathcal{X}$ and let $Y \in \mathcal{Y}(B_i)$. Y is incompatible with G_i^* if and only if there are $u, v \in B_i$ with $uv \in E(G_i^*) \setminus M^*$ and $Y(u) = "N"$ and $Y(v) \neq "\times"$.*

Proof. "⇒":

$uv \in E(G_i^*) \setminus M^*, Y(u) = "N"$ and $Y(v) \neq "\times"$. In this case, Definition 8(i) cannot be verified, then there is not set X with Y as signature.

"⇐": We show that any other signature Y can have a set X eligible with respect to the tuple (Y, B_i). Let $S = \{u \mid Y(u) = "\times"\} \cup V(G_i^*) \setminus B_i$. Let $u \in B_i$ and $uv \in E(G_i^*) \setminus M^*$. Suppose that $Y(u) = "N"$. If $v \notin B_i$, then $v \in X$, otherwise $Y(v) = "\times"$ and then $v \in X$. Thus, the conditions of Definition 8 are verified and Y is the signature of X in B_i. For each matching edge uv, $d_{G^*}(u) = 1$, $d_{G^*}(v) = 1$ or $\{u, v\} \cap X \neq \varnothing$, then uv is clean and then X is a solution of Semi-Brutal Cut in G_i^*. Thus, X is eligible with respect to (Y, B_i).

Semantics: *Let $Y : V(G_i^*) \to \{"\times","N","\varnothing"\}$. A table entry $[Y]_i$ is some minimum-score solution X that is eligible with respect to (Y, B_i) (and $[Y]_i = \bot$ if no such X exists).*

We set $\text{score}(\bot) = \infty$ and $\bot \cup X = \bot$, for any set X.

The Algorithm. Let (G^*, M^*, ω, m) be a solution graph. We compute a M^*-preserving tree decomposition $(\mathcal{X}, T)$ of G^* as described previously. We then traverse $(\mathcal{X}, T)$ from the leaf bags to the root X_r. We compute the table entry for each signature $Y \in \mathcal{Y}(B_i)$ of each bag B_i. Then, we obtain the minimum solution for (G^*, M^*, ω, m) from $[Y]_r$. Let B_j and B_ℓ the children of B_i (if they exist). We compute $[Y]_i$ depending on the type of the bag B_i:

leaf bag: Since $B_i = \varnothing$, the only table entry is $[\varnothing]_i$ and we set $[\varnothing]_i = \varnothing$.

introduce $uv \in M^*$ **bag:** We apply the following routine:

1. First consider that uv is isolated. We copy the table entries of the child B_j and complete them such that all signatures in $\mathcal{Y}(B_i)$ are instantiated:

$$[Y]_i = \operatorname*{argmin}_{Y \in \mathcal{Y}(B_j)} \operatorname*{argmin}_{Y' \in \mathcal{Y}(\{u,v\})} \{\text{score}([Y]_j \cup \mathcal{T}(Y'))\}.$$

2. Then, we introduce successively the non-matching edges incident to uv. If a signature is incompatible with G_i^*, then we set its table entry to $\bot$. Let E' be the set of incident edges to the matching edge uv. For each $xx' \in E'$ and all $Y \in \mathcal{Y}(B_i)$, we set $[Y]_i = \bot$ if
   - if $Y(x) = "N"$ and $Y(x') \neq "\times"$ (Lemma 2),
   - if $Y(x') = "N"$ and $Y(x) \neq "\times"$ (Lemma 2).

join bag: For all $Y \in \mathcal{Y}(B_i)$, we set $[Y]_i = [Y]_j \cup [Y]_\ell$.

forget $uv \in M^*$ bag: For all $Y \in \mathcal{Y}(B_i)$, we set

$[Y]_i = \text{argmin}(\{\text{score}([Y']_j) \mid Y' \in \mathcal{Y}(B_j) \wedge \exists_{Y'' \in \mathcal{Y}(\{u,v\})} Y' = Y \cup Y''\})$.

Lemma 3. *The described algorithm is correct, that is, the computed value of $[Y]_i$ corresponds to the semantics.*

Proof. The proof is by induction on the height of B_i in the tree decomposition. In the induction base, B_i is a leaf of $(\mathcal{X}, T)$ and $B_i = \varnothing$. Thus, the only table entry is $[\varnothing]_i$ and since $\varnothing$ is the only solution, $[\varnothing]_i = \varnothing$ corresponds to the defined semantics. For the induction step, we distinguish the possible bag types of B_i with children B_j and B_ℓ ($j = \ell$ if B_i is not a join bag).

Introduce matching edge uv bag: We prove that each step of the algorithm is correct. We introduce the non-matching edges only in the second step. In the first step, we consider that uv is isolated in G_i^*.

1. When an isolated matching edge uv is introduced, if X is a solution of G_j^*, then each vertex set X' such that $X \subseteq X'$ is also a soluton G_i^*. For each $Y \in \mathcal{Y}(B_j)$, since $[Y]_j$ is the solution with a minimum score and with the signature Y in B_j, for each $Y' \in \mathcal{Y}(\{u,v\})$ we have that $[Y]_j \cup \mathcal{T}(Y')$ is a minimum-score solution with signature $Y \cup Y'$ in B_i. Thus, $[Y \cup Y']_i = [Y]_j \cup \mathcal{T}(Y')$ corresponds to the semantics.
2. We now introduce successively each non-matching edge. Let E' be the set of incident edges to the matching edge uv. For each $xx' \in E'$ and each $Y \in \mathcal{Y}(B_i)$,
   - if $Y(x) = "N" \wedge Y(x') \neq "\times"$ or $Y(x') = "N" \wedge Y(x) \neq "\times"$, then, by Lemma 2, there is no solution eligible with respect to the tuple (Y, B_i) and, thus $[Y]_i = \bot$ is valid.

   If $[Y]_i \neq \bot$, then by Lemma 2, there is a solution X eligible with respect to the tuple (Y, B_i). We show that $[Y]_i$ contains a minimum-score solution. Suppose that there is a solution X' with the signature Y in B_i such that $\text{score}(X') < \text{score}([Y]_i)$. Let $Y' \in \mathcal{Y}(B_j)$ and $Y'' \in \mathcal{Y}(\{u,v\})$ such that $Y = Y' \cup Y''$. Since $X' \cap \mathcal{T}(Y) = [Y]_i \cap \mathcal{T}(Y)$, we have $\text{score}(X' \setminus (\mathcal{T}(Y) \cap \{u,v\})) < \text{score}[Y']_j$, contradicting the induction hypothesis.

Join bag: Let $Y \in \mathcal{Y}(B_i)$ and $u \in B_i$. Notice that $G_j^* \cap G_\ell^* = B_i$. Thus, a cut in the solution $[Y]_j \setminus B_i$ can not remove an edge incident to a vertex of $G_\ell^* \setminus B_i$. Without loss of generality, if there is no solution eligible for the pair (Y, B_j) (that is, $[Y]_j = \bot$), then there is no solution eligible for (Y, B_i). If $[Y]_j$ cleans all matching edges in G_j^* and $[Y]_\ell$ cleans all matching edges in G_ℓ^*, then $[Y]_j \cup [Y]_\ell$ cleans all matching edges in $G_j^* \cup G_\ell^*$. Thus, $[Y]_j \cup [Y]_\ell$ is eligible for (Y, B_i). Suppose that there is a solution X' eligible with respect for (Y, B_i) such that $\text{score}(X') < \text{score}([Y]_j \cup [Y]_\ell)$ in G_i^*. Then, score$(X') \cap V(G^*_j) <$ score$([Y]_j)$ in G^*_j or score$(X') \cap V(G^*_\ell) <$ score$([Y]_\ell)$ in G^*_ℓ. Both cases contradict the induction hypothesis. Thus, $[Y]_i \cup [Y]_j$ has a minimum score and $[Y]_i = [Y]_j \cup [Y]_\ell$.

Forget matching edge uv **bag:** Since $G^*_i = G^*_j$, each solution in G^*_j is also a solution in G^*_i. Since, for each $Y \in \mathcal{Y}(B_i)$, all solutions of $S = \{[Y']_j \mid \forall Y'' \in \mathcal{Y}(\{u,v\}), Y' = Y \cup Y''\}$ have the signature Y in B_i, we store a minimum-score solution of X in the table entry $[Y]_i$. Thus, $[Y]_i = \text{argmin}(\{\text{score}([Y']_j) \mid Y' \in \mathcal{Y}(X_j) \wedge \exists Y'' \in \mathcal{Y}(\{u,v\}), Y' = Y \cup Y''\})$ is valid. □

In each bag B_i, we have to iterate over all signatures in $\mathcal{Y}(B_i)$. The number of possible values for an ambiguous edge is equal to two under the weight score and to five under the cut score. Thus, the number of signatures in a bag containing tw matching edges is equal to 2^{tw} under the weight score and to 5^{tw} under the cut score. Since the number of bags depends on the number of non-matching edges, we obtain a complexity of $O(2^{tw} \cdot |E(G')|)$ under the weight score and a complexity of $O(5^{tw} \cdot |E(G')|)$ under the cut score. To obtain an optimal solution, we just have to take the value of $[\varnothing]_r$ computed by the algorithm.

Corollary 1. *Given a M^*-preserving nice tree decomposition with width tw,* Semi-Brutal Cut *can be solved in $O(2^{tw} \cdot |E(G^*)|)$ time under the weight score and in $O(5^{tw} \cdot |E(G^*)|)$ time under the cut score.*

Optimization As no non-ambiguous matching edge will contain a cut, we can remove these matching edges from the graph before computing the tree decomposition, yielding a reduction of the treewidth. However, we must ensure that each vertex stores its adjacency to a removed matching edge.

6 Experiments

The contribution of the paper being mainly theoretical, we propose implementation and tests on real instances. In order to compare the performance of both algorithms, we tested them on datasets already used in [14].We can observe that selected instances have a small treewidth. A real instance of SBC is generated from a collection of alternating paths and alternating cycles, thus we may think that such instance has a small treewidth. Our implementation of the tree decomposition based algorithm relies on the HTD library [1] for tree decomposition construction. We use ILOG CPLEX to provide a solution to our integer linear programming formulation. We compare results for both scores, statistics on produced solutions are presented in Table 1. Additional statistics on instances are given in Table 2. We can see that the tree decomposition algorithm is faster under the weight score, which can be explained by the difference of the theoretical complexity. For the cut score, the dynamic programming is slightly faster than the ILP with one exception for the *anopheles* genome. Since the real instances seem to have a small treewidth and the tree decomposition algorithm uses more the internal structure of the problem, we may think that it remains faster than the ILP.

Table 1. Results statistics. "ILP" and "Tree Dec." columns indicate the execution times, in seconds.

	Tree	Cut Score			Weight Score		
data	-width	Score	ILP	Tree Dec.	Score	ILP	Tree Dec.
anopheles	3	1093	4.63	5.10	1387	4.76	4.22
anthrax	2	12	0.42	0.32	17	0.41	0.31
gloeobacter	2	39	0.44	0.36	67	0.46	0.36
lactobacillus	2	13	0.19	0.15	18	0.19	0.14
pandora	1	5	0.25	0.19	6	0.25	0.18
pseudomonas	2	36	0.54	0.42	51	0.53	0.42
rice	2	3	0.01	0.00	3	0.01	0.00
sacchr3	2	3	0.03	0.02	5	0.03	0.02
sacchr12	4	12	0.10	0.07	18	0.09	0.07

7 Conclusion

In this paper, we present a novel point of view on a problem dedicated to the production of genomic sequences. The previous exploration of the frontier between tractable and hard cases did not provide a satisfactory polynomial-time algorithm and, thus, we explore here two possible solutions: The first is to position the problem relative to the PLS class, aiming to decide whether local search is easier than global search. The second is to consider natural exact methods. In this context, we studied and implemented a simple and efficient ILP and a tree-decomposition based method, yielding an FPT algorithm with respect to the treewidth of the input graph. Interesting open questions include the existence of polynomial-time approximation algorithms, and whether alternative tools, such as color coding or kernel techniques, allow designing more efficient FPT algorithms. As a more practical perspective, we intend to perform further tests on these algorithms and previous ones, to explore the ability of each method to perform well on various kinds of genomes.

References

[1] M. Abseher, N. Musliu, and S. Woltran. htd - A free, open-source framework for (customized) tree decompositions and beyond. In *Proc. of the 14th International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR 2017)*, volume 10335 of *LNCS*, pages 376–386. Springer, 2017.

[2] M. A. Biscotti, E. Olmo, and J. S. Heslop-Harrison. Repetitive DNA in eukaryotic genomes. *Chromosome Res.*, 23(3):415–420, Sep 2015.

[3] P. Bongartz. Resolving repeat families with long reads. *BMC Bioinformatics*, 20 (232), 05 2019. ISSN 1471-2105. doi: 10.1186/s12859-019-2807-4.

[4] A. Chateau and R. Giroudeau. A complexity and approximation framework for the maximization scaffolding problem. *Theoretical Computer Science*, 595:92–106, 2015.

[5] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.

[6] T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[7] M. Krentel. On finding and verifying locally optimal solutions. *SIAM Journal on Computing*, 19(4):742–749, 1990.

[8] I. Mandric, J. Lindsay, I. I. Măndoiu, and A. Zelikovsky. Scaffolding algorithms. In I. Măndoiu and A. Zelikovsky, editors, *Computational Methods for Next Generation Sequencing Data Analysis*, chapter 5, pages 107–132. Wiley, 2016.

[9] A. A. Schäffer and M. Yannakakis. Simple local search problems that are hard to solve. *SIAM J. Comput.*, 20(1):56–87, 1991.

[10] D. Tabary, T. Davot, A. Chateau, R. Giroudeau, and M. Weller. New results about the linearization of scaffolds sharing repeated contigs. In *COCOA 2018*, Lecture Notes in Computer Science, pages 94–107. Springer, 2018.

[11] H. Tang. Genome assembly, rearrangement, and repeats. *Chemical Reviews*, 107 (8):3391–3406, 2007.

[12] A. I. Tomescu, T. Gagie, A. Popa, R. Rizzi, A. Kuosmanen, and V. Mäkinen. Explaining a weighted DAG with few paths for solving genome-guided multi-assembly. *IEEE/ACM Trans. Comp. Biology Bioinform.*, 12(6):1345–1354, 2015.

[13] A. Ummat and A. Bashir. Resolving complex tandem repeats with long reads. *Bioinformatics*, 30(24):3491–3498, 07 2014. ISSN 1367-4803. doi: 10.1093/bioinformatics/btu437. URL https://doi.org/10.1093/bioinformatics/btu437.

[14] M. Weller, A. Chateau, and R. Giroudeau. Exact approaches for scaffolding. *BMC Bioinformatics*, 16(Suppl 14):S2, 2015.

[15] M. Weller, A. Chateau, and R. Giroudeau. On the linearization of scaffolds sharing repeated contigs. In *Combinatorial Optimization and Applications - 11th International Conference, COCOA 2017, Shanghai, China, December 16-18, 2017, Proceedings, Part II*, pages 509–517, 2017.

[16] M. Weller, A. Chateau, C. Dallard, and R. Giroudeau. Scaffolding problems revisited: Complexity, approximation and fixed parameter tractable algorithms, and some special cases. *Algorithmica*, 80(6):1771–1803, 2018.

[17] M. Weller, A. Chateau, R. Giroudeau, and M. Poss. Scaffolding with repeated contigs using flow formulations. 2018.

A Appendix

A.1 Supplementary materials

Table 2. Sequences selected for experiments

data	#AE[1]	#NAE[2]	total weight	avg. deg.[3]	max / min deg.	treewidth
anopheles	1523	7695	14937	2.43598	6 / 2	3
anthrax	13	260	329	2.65385	4 / 2	2
gloeobacter	44	432	694	2.84091	6 / 2	2
lactobacillus	15	135	225	2.63333	5 / 2	2
pandora	5	183	210	2.5	4 / 2	1
pseudomonas	47	413	650	2.59574	5 / 2	2
rice	6	9	29	2.08333	3 / 2	2
sacchr3	5	25	54	2.7	4 / 2	2
sacchr12	23	74	190	2.43478	4 / 2	4

1. ambiguous edges 2. non-ambiguous edges 3. average degree of extremities of amb. paths

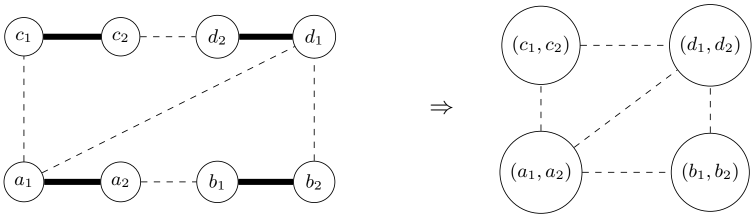

Fig. 4. Application of the contraction. **Left:** a solution graph G^*, the matching edges are in bold. **Right:** graph resulting of the contraction of the matching edges in G^*.

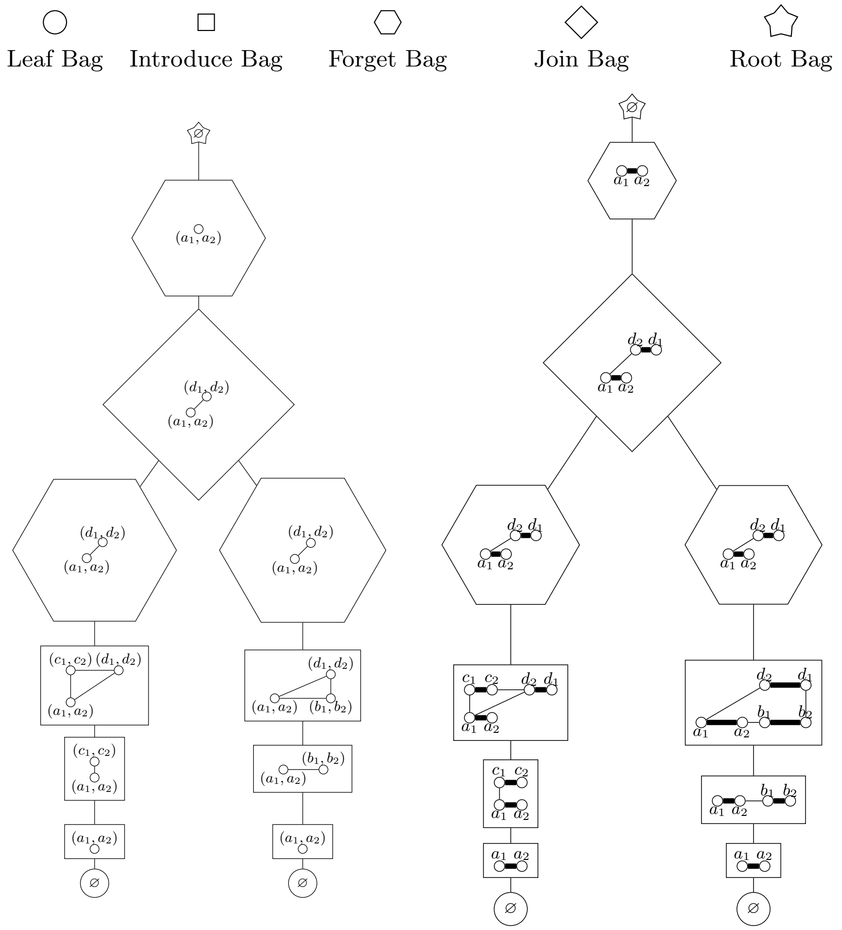

Fig. 5. Nice tree decomposition of the graph given in Figure 4. **Left:** Nice tree decomposition of the graph after applying the contraction operation. **Right:** Same nice tree decomposition after replacing the vertices by their corresponding matching edges.