New Polynomial-Time Algorithm around the Scaffolding Problem

Tom Davot, Annie Chateau, Rodolphe Giroudeau, Mathias Weller

New Polynomial-Time Algorithm around the Scaffolding Problem

Tom Davot[1], Annie Chateau[1,2], Rodolphe Giroudeau[1], Mathias Weller[3]

[1] LIRMM - CNRS UMR 5506 - Montpellier, France
[2] IBC - Montpellier, France
[3] LIGM, Bât Copernic - Champs-s/-Marne, France
{tom.davot,annie.chateau,rodolphe.giroudeau}@lirmm.fr,
mathias.weller@u-pem.fr

Abstract. We describe in this paper an approximation algorithm for the scaffolding problem, which is part of genome inference in bioinformatics. The aim of the problem is to find a maximum weighted collection of disjoint alternating cycles and paths covering a particular graph called scaffold graph. The problem is known to be NP-complete, and we describe further result concerning a special class of graphs aiming to be close to real instances. The described algorithm is the first polynomial-time approximation algorithm designed for this problem on non-complete graphs.

1 Introduction

Motivation. We are interested here in an algorithmic problem occuring in the production of genomes. Genomes are usually obtained by sequencing, wich produces a set of *reads* whose length and quality depend on the sequencing technology. It is commonly known that short reads (typically hundreds of base pairs), produced by second generation sequencing technology (Illumina), are of better quality than long reads (thousands of base pairs), produced by third generation sequencing technologies (PacBio or Oxford Nanopore) [6]. Those reads are then assembled using a variety of tools, the most recent integrating very efficient hybrid strategies using both short and long reads [7]. However, databases are full of "old genomes", produced before the development of third generation sequencing, and "hard genomes" that escape sequencing in good conditions. In fact, most of the genomes in databases consist as huge sets of chunks of sequences, called *contigs*. These sets contain far more contigs than the real number of the chromosomes of the corresponding organisms. Such fragmentation is observed even for well-studied genomes. To the natural question "how to reduce this fragmentation?", technological progress and costly re-sequencing is not the only answer and computational exploitation of already available sequencing data is possible.

Scaffolding Problem. We focus here on the contig scaffolding problem which, given a set of contigs, asks to infer the order and the orientation of the contigs along the target genome, using a set of possibly inconsistent pairing information. This information could be provided, for instance, by paired-end reads whose

two ends map to distinct contigs. Formally, it is possible to extract from this information a set of relationships between the contigs, that may be inconsistent. A survey on recent methods is available in [5]. The problem that we model here is more general than those presented in the literature, and therefore allows adaptation towards realistic modelization. We study the scaffolding problem as an optimization problem in a graph called *scaffold graph*, obtained by mapping of paired-end reads on *de novo* contigs. However, the present formulation is not limited to this aspect, and may also consider other sources of information. We consider that the scaffolding may obey genomic structural constraints, like a fixed number of linear and circular chromosomes. In the past, we presented preliminary results about the complexity of this problem and a first polynomial-time approximation algorithm on cliques [1]. Those results were extended and completed by another polynomial-time approximation algorithm [2], and by a randomized approach [3]. Exact approaches have been explored [9], leading to study sparse cases [8]. The contribution of the present paper is a continuation of [4, 10], where special classes of graphs has been studied, from sparse to very dense. Real instances are very sparse, but show some dense regions. Hence, we are interested in graphs built from cliques separated by bridges (i.e. edges whose removal disconnects the graph).

2 Notation and Problem Description

In this section, we formally define the Scaffolding problem. For a graph G, we denote by $V(G)$ and $E(G)$ the set of vertices and edges of G, respectively. A *scaffold graph* (G^*, M^*, ω) is a simple loopless graph G^* with a perfect matching M^* and a weight function ω on the non-matching edges. The matching M^* respresents the set of contigs and the function ω represents the confidence that two contigs occur consecutively in the genomic sequence. An *alternating path* (resp. *alternating cycle*) is a path (resp. cycle) such that its edges alternatively belong to M^* or not. The extremal edges of an alternating path must be in M^*. The Scaffolding problem, whose decision version is NP-complete on general graphs [2], is defined as follows:

(σ_p, σ_c)-Scaffolding (SCA)
Input: a scaffold graph (G^*, M^*, ω) and integers $\sigma_\mathrm{p}, \sigma_\mathrm{c}$.
Task: Find a collection S of σ_p alternating paths and σ_c alternating cycles maximizing $\sum_{e \in S \setminus M^*} \omega(e)$

The two integers σ_p and σ_c are used to model the genomic structure by representing the number of linear and circular chromosmes, respectively. Let S be a partial solution of Scaffolding, the *cardinality* of S is the number of alternating paths and cycles which compose S. We denote by $\sigma_p(S)$ and $\sigma_c(S)$ the number of alternating paths and alternating cycles of S, respectively. The approximation algorithm used in the following is described in Algorithm 1. It is known that this algorithm produces a solution for Scaffolding with an approximation ratio of three in complete graphs [2].

Algorithm 1: Polynomial-time approximation algorithm for (σ_p, σ_c)-Scaffolding Problem.

Data: A scaffold graph (G^*, M^*, ω), two integers σ_p and σ_c.
Result: A collection of σ_p alternating paths and σ_c alternating cycles or "False" if no such collection exists.

```
// Initialization step
```
$S \leftarrow M^*$;
$E \leftarrow E \setminus M^*$;
sort E by decreasing order of weight;
if *not* $Feasibility((G^*, M^*), S, \sigma_p, \sigma_c)$ **then return** *False*;
```
// Main loop
```
while $E \neq \varnothing$ **do**
    Let $e = \{u, v\}$ be the first element in the ordered-list E;
    $E \leftarrow E \setminus e$;
    if $Feasiblity((G^*, M^*), S \cup \{e\}, \sigma_p, \sigma_c)$ **then**
        $R \leftarrow$ set of edges of E incident to e;
        $S \leftarrow S \cup \{e\}$;
        $E \leftarrow E \setminus R$;
return S;

To adapt Algorithm 1 for another class of graphs, we need to provide a dedicated feasibility function. This function, given a partial solution S, indicates if it is possible to build a solution to Scaffolding with the remaining edges. In this paper, we use Algorithm 1 on a particular class of graphs defined as follows:

Definition 1. *A* connected cluster graph G *is a graph which admits a decomposition of its edges* $E(G) = E' \cup B$ *such that the subgraph induced by* E' *is a disjoint union of cliques and each edge* $e \in B$ *is a bridge of* G.

An example of a connected cluster graph is given in Figure 1. Let G be a connected cluster graph, for sake of simplicity, we designate by *clique* a connected component of the subgraph induced by E' and we denote by $CC(G)$ the set of cliques of G.

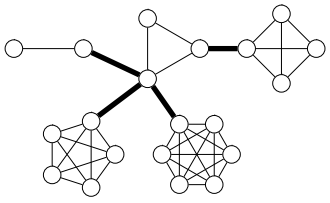

Fig. 1. Example of connected cluster graph. The bridge edges are bold.

As the structure of a connected cluster graph is close to a tree (that is, shrinking each clique of G^* into a single vertex leads to a tree), we use a similar vocabulary: a *rooted* connected cluster graph is a a connected cluster graph where a clique r is designated as a *root* and in that case, the *parent* of a clique is the clique connected to it on the path to r. A *child* of a clique c is the clique of which c is the parent. A vertex v of a clique c is a *door* of c if a child of c is adjacent to v. The *upper door* of c is the vertex adjacent to the parent of c. In the following, we will focus on scaffold graphs (G^*, M^*, ω) such that G^* is a connected cluster graph and $M^* \cap B = \varnothing$. Since the decision version of Scaffolding is NP-complete on cliques [2], it is also NP-complete on connected cluster graphs.

3 Feasibility

In this section, we present an algorithm to determine if it is possible to construct a solution of SCAFFOLDING in a connected cluster graph. Its principle is to construct and assemble some partial solutions in a bottom-up traversal of the connected cluster graph. Instead of storing the feasible solutions, we store their cardinalities.

Operations. Let G_1 and G_2 be two edge-disjoint subgraphs. We can build a solution in the graph induced by $V(G_1) \cup V(G_2)$, from a solution in G_1 and a solution in G_2, using four operations.

Definition 2. *Let G_1 and G_2 be edge-disjoint subgraphs of G^*. Let S_1 and S_2 be solutions of G_1 and G_2, respectively. Let S be a solution of $G^*[V(G_1) \cup V(G_2)]$. S is a* composition *of S_1 and S_2 if exactly one of the following operations occurs:*

Merger: *merge a path of S_1 with a path of S_2 in S.*

Closing: *close a path of S_1 and a path of S_2 into an alternating cycle in S.*

Absorption: *replace a non-matching edge vv' of S_2 by an alternating path of S_1, that is, $S = S_1 \cup (S_2 \setminus \{vv'\}) \cup \{uv, u'v'\}$ where u and u' are extremities of a u-u'-path in S_1. We call vv'* absorbent.

Juxtaposition: *S is the disjoint union of S_1 and S_2 and none of the previous operations are performed.*

To implement these operations, we add edges of $E(G^*) \setminus (E(G_1) \cup E(G_2))$ to S.

Note that a composition of two solutions does not always exist, except for the juxtaposition operation. In the algorithm, we manipulate sets of solutions instead of solutions. Thus, we can create a new set of solutions if all the solutions of the two input sets are used in the resulting set.

Definition 3. *Let G_1 and G_2 be two edge-disjoint subgraphs of G^* and let $\mathcal{S}_1$ and $\mathcal{S}_2$ be sets of solutions of subgraphs G_1 and G_2, respectively. Then, we call the set $\mathcal{S} = \{S \mid \exists S_1 \in \mathcal{S}_1, \exists S_2 \in \mathcal{S}_2$ s.t. S is a composition of S_1 and $S_2\}$ the* complete composition *of $\mathcal{S}_1$ and $\mathcal{S}_2$.*

To ensure the possibility of building a complete composition from two sets of solutions, it is useful to characterize a solution according to the operations we can perform on it. Thus, given two subgraphs G_1 and G_2, we define four properties on a solution S according to the operations on S.

Definition 4. *Let G and G' be two vertex-disjoint subgraphs of G^* and let S be a feasible solution of* SCAFFOLDING *for (G, M^*, ω).*

1. *We call S* closeable *if S contains an alternating u-v-path and there is an alternating[4] u-v-path in $G' \cup \{u, v\}$.*
2. *We call S* extensible *by G' if S contains a vertex v such that v is an extremity of an alternating path and v has a neighbor in G' .*
3. *We call S* frozen *to G' if S is not extensible.*
4. *We call S* absorbent *to G' if S contains a non-matching edge uv and G' contains a matching edge $u'v'$ such that $uu', vv' \in E(G^*)$.*

[4] we use here "alternating" in an abusive manner, meaning alternating matching edges and non-matching edges, beginning and ending with non-matching edges

For simplicity, we sometimes omit to precise G' and in this case $G' = G^* - V(G)$. Note that all closeable solutions are also extensible. If a solution S is closeable by a subgraph G', then we can close an alternating path of S into an alternating cycle by adding some edges of G'. If a solution S is extensible by a subgraph G', then we can add some edges of G' in an extremity of an alternating path of S without changing the cardinality of the solution. Finally, if a solution S is absorbent to a subgraph G', then we can replace an absorbent edge of S by a path of length three without changing the cardinality of S. An example of the different operations of Definition 4 is given in Figure 2.

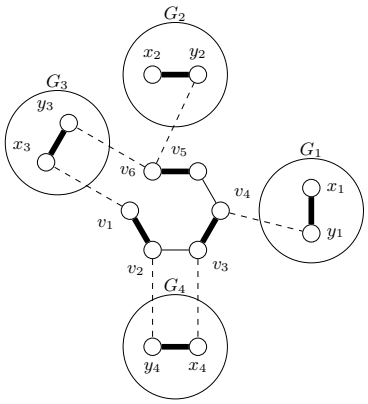

Fig. 2. The solution S is composed of a single alternating path $\{v_1, \ldots, v_6\}$. S is closeable by subgraph $G_3 = \{x_3, y_3\}$: we can close the alternating path of S into an alternating cycle by adding the edges v_1x_3, x_3y_3 and y_3v_6. S is extensible by subgraph $G_2 = \{x_2, y_2\}$: we can extend the alternating path of S by adding the edges v_6y_2 and y_2x_2 without changing the number of paths in S. S is absorbent to $G_4 = \{x_4, y_4\}$: we can replace the edge v_2v_3 of S by the edges v_2y_4, y_4x_4 and x_4v_3 without changing the number of paths in S. S is frozen to $G_1 = \{x_1, y_1\}$.

Semantics. Since the number of possible solutions can be exponential, we just store the possible cardinalities in the table entries, which is sufficient to answer the question of feasibility. We recall that, if $X, Y \subseteq \mathbb{N}$ are two sets of integers, then the sum of X and Y is defined as $X + Y = \{x + y \mid x \in X, y \in Y\}$. Note that $X + \varnothing = \varnothing$.

Definition 5. *Let $\mathcal{S}$ be a set of solutions and $i, j \in \mathbb{N}$. Then, j is called* eligible *with respect to $(\mathcal{S}, i)$ if there is a solution $S \in \mathcal{S}$ containing i alternating cycles and j alternating paths.*

Our dynamic programming table has the following semantics.

Semantics. *Let $\mathcal{S}$ be a set of solutions and $i \in \mathbb{N}$. A table entry $[\mathcal{S}, i]$ is the set of all integers eligible with respect to the tuple $(\mathcal{S}, i)$. More formally, letting $\mathcal{S}_i = \{S \mid S \in \mathcal{S} \wedge \sigma_c(S) = i\}$, we define $[\mathcal{S}, i] = \bigcup_{S \in \mathcal{S}_i} \{\sigma_p(S)\}$.*

Let us highlight three particular values of $[\mathcal{S}, i]$. For $\mathcal{S} = \{\varnothing\}$, we have $[\{\varnothing\}, 0] = 0$ and, for each $i > 0$, we have $[\{\varnothing\}, i] = \varnothing$. For an alternating path p, we have $[\{p\}, 0] = 1$ and $[\{p\}, i] = \varnothing$ for each $i > 0$. Finally, for an alternating cycle c, we have $[\{c\}, 1] = 0$ and $[\{c\}, i] = \varnothing$ for each $i \neq 1$. For simplicity, we denote by $[\mathcal{S}]$ the vector $([\mathcal{S}, 0], \ldots, [\mathcal{S}, \sigma_c])$ and, for any operator $\diamond$ and any sets $\mathcal{S}_1$ and $\mathcal{S}_2$

of solutions, we define $[\mathcal{S}_1] \diamond [\mathcal{S}_2]$ as component-wise $\diamond$, that is, $[\mathcal{S}_1, i] \diamond [\mathcal{S}_2, i]$ for each $i \in [0, \sigma_c]$.

Lemma 1. *Let G_1 and G_2 be two vertex-disjoint subgraphs of G^* and let $\mathcal{S}_1$ and $\mathcal{S}_2$ be sets of solutions of subgraphs G_1 and G_2, respectively. Let $\mathcal{S}$ be a set of solutions of $G^*[V(G_1) \cup V(G_2)]$ such that $\mathcal{S}$ is a complete composition of $\mathcal{S}_1$ and $\mathcal{S}_2$.*

1. *If $\mathcal{S}$ is the set of solutions composed with a merger operation, then $\forall i, j, [\mathcal{S}, i + j] = [\mathcal{S}_1, i] + [\mathcal{S}_2, j] + \{-1\}$.*
2. *If $\mathcal{S}$ is the set of solutions composed with a closing operation, then $\forall i, j, [\mathcal{S}, i + j + 1] = [\mathcal{S}_1, i] + [\mathcal{S}_2, j] + \{-2\}$.*
3. *If $\mathcal{S}$ is the set of solutions composed with an absorption operation, then $\forall i, j, [\mathcal{S}, i + j + 1] = [\mathcal{S}_1, i] + [\mathcal{S}_2, j] + \{-1\}$.*
4. *If $\mathcal{S}$ is the set of solutions composed with a juxtaposition operation, then $\forall i, j, [\mathcal{S}, i + j + 1] = [\mathcal{S}_1, i] + [\mathcal{S}_2, j]$.*

Proof. Let $S \in \mathcal{S}$, we denote by S_1 and S_2 the solutions of $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively, such that S is composed by S_1 and S_2. Now we prove the four statements of the lemma.

1. $\forall S \in \mathcal{S}$, since S_1 and S_2 have a common alternating path in S, we have $\sigma_p(S) = \sigma_p(S_1) + \sigma_p(S_2) - 1$ and since no cycle is formed, $\sigma_c(S) = \sigma_c(S_1) + \sigma_c(S_2)$. Thus,$\forall i, j, [\mathcal{S}, i + j] = [\mathcal{S}_1, i] + [\mathcal{S}_2, j] + \{-1\}$.
2. $\forall S \in \mathcal{S}$, since one path of S_1 and one path of S_2 are closed into a single alternating cycle, we have $\sigma_p(S) = \sigma_p(S_1) + \sigma_p(S_2) - 2$ and $\sigma_c(S) = \sigma_c(S_1) + \sigma_c(S_2) + 1$. Thus, $\forall i, j, [\mathcal{S}, i + j + 1] = [\mathcal{S}_1, i] + [\mathcal{S}_2, j] + \{-2\}$.
3. $\forall S \in \mathcal{S}$, since S_1 has an alternating path that is "absorbed" into a connected component of S_2, we have $\sigma_p(S) = \sigma_p(S_1) + \sigma_p(S_2) - 1$ and since no cycle is formed, $\sigma_c(S) = \sigma_c(S_1) + \sigma_c(S_2)$. Thus,$\forall i, j, [\mathcal{S}, i + j] = [\mathcal{S}_1, i] + [\mathcal{S}_2, j] + \{-1\}$.
4. $\forall S \in \mathcal{S}$ since all paths and cycles of S_1 and S_2 are present in S, we have $\sigma_p(S) = \sigma_p(S_1) + \sigma_p(S_2) - 1$ and since no cycle is formed, $\sigma_c(S) = \sigma_c(S_1) + \sigma_c(S_2)$. Thus,$\forall i, j, [\mathcal{S}, i + j] = [\mathcal{S}_1, i] + [\mathcal{S}_2, j] + \{-1\}$.

We use Lemma 1 to define four applications $juxtapose$, $merge_t$, $absorb$, $close_t$ which provide table entries for complete compositions "composed" with a juxtaposition, merger, absorption or closing operation, respectively. Although Lemma 1 is defined for two sets, we use a generalized version which can take as parameters more than two sets. The functions $merge_t$ and $close_t$ have a parameter t which indicates the number of paths merged or closed during the operation. For example, if we have three sets S_1, S_2, and S_3 and it is possible to construct a single alternating path in the resulting composition by taking one alternating path in each set, then we use the function $merge_3(\{S_1\}, \{S_2\}, \{S_3\})$. In addition, it is sometimes possible to close a single alternating path into an alternating cycle and in that case the function $close_1$ is used. The four applications are defined in Algorithm 7, Algorithm 8 and Algorithm 9 (in appendix). However, we must ensure that the associated operation is feasible before using one this application.

The Algorithm. We now present a method to provide the feasibility function needed by Algorithm 1. We suppose that a partial solution S is given. Let c be

a clique of G^* and let S' be the intersection of S and c. An *alternating element* of c is either an alternating cycle of S' or an alternating path of S'. We traverse different types of subgraphs defined in the following way:

- Let $v \in V(G^*)$, let $C(v)$ be the set of children adjacent to v (possibly empty). The subgraph $G^*(v)$ is the union of v and all branches incident to v. Formally, $G^*(v) = G^*[\{v\} \cup \bigcup_{c \in C(v)} V(G^*(c))]$.
- Let e be an alternating element, the subgraph $G^*(e)$ is the union of e and all children incident to one of its vertices. Formally, $G^*(e) = G^*[\bigcup_{v \in e} V(G^*(v))]$.
- Let c be a clique of G^*, and let dd' be the matching edge of c incident to the upper door of c. Let $c' = c \setminus dd'$ be the *subclique* of c. For all $x \in \{c, c'\}$, the subgraph $G^*(x)$ is the union of x and all children incident to a vertex of x. Formally, $G^*(x) = G^*[\bigcup_{e \in M^*(x)} V(G^*(e))]$.

For each traversed subgraph G', we use four different sets of solutions distinguishing solutions according to their properties.

Definition 6. *Let S be a partial solution of G^*. Let x be a vertex, a partial path, a subclique or clique of G^* and let S' be a solution of the subgraph $G^*(x)$.*

- $S \in \mathcal{C}(x) \Leftrightarrow S'$ *is closeable and* $S \cap G^*(x) \in S'$.
- $S \in \mathcal{P}(x) \Leftrightarrow S \notin \mathcal{C}(x)$ *and S is extensible and* $S \cap G^*(x) \in S'$.
- $S \in \mathcal{A}(x) \Leftrightarrow S$ *is frozen and absorbent and* $S \cap G^*(x) \in S'$.
- $S \in \mathcal{F}(x) \Leftrightarrow S \notin \mathcal{A}(x)$ *and S is frozen and* $S \cap G^*(x) \in S'$.

The next paragraphs are dedicated to describing the algorithms to calculate the table entries for the four types of subgraphs described above.

Vertex. Let v be a vertex of G^*. We show in this part how to compute the table entries for the sets $\mathcal{F}(v)$ and $\mathcal{P}(v)$. Note that, since the edge between $G^*(v)$ and its parent is a bridge, the sets $\mathcal{C}(v)$ and $\mathcal{A}(v)$ are empty. Any solution S' of $G^*(v)$ can have at most one incident edge to v. If no edge of $S \cap G^*(v)$ is incident to v, the idea is to construct the table entries by merging successively the table entries of the children incident to v. For that, we use at each step an intermediate graph G_i. Let V_i be the union of the i first children of v. G_i is the subgraph of G^* induced by v and all vertices in V_i. Otherwise, if one edge of $S \cap G^*(v)$ is incident to v, then any solution containing S belongs to $\mathcal{P}(v)$.

Lemma 2. *For any vertex v, the values of the table entries provided by Algorithm 2 are correct for the set $\mathcal{F}(v)$ and $\mathcal{P}(v)$.*

Proof. First, if there is no child linked to v, then $G^*(v)$ is constituted by the single vertex v. In that case, the only solution is that containing zero alternating cycle and path and this solution is frozen. Thus, the initial values given to $[\mathcal{F}(v)]$ and $[\mathcal{P}(v)]$ in the initialization step (i.e. lines 1 to 2) are correct. If it exists an edge uv of $E(G^*(v)) \cap S$ incident to v, then let c_u be the clique containing u. A solution of $G^*(v)$ is necessarily composed by the juxtaposition of an extensible solution of c_u and any solution of the other children and this solution is extensible. Thus, the assignments lines 9 and 12 are correct. Now suppose that uv does not exist. Let $c_t \in C(v)$ be the clique considered at step t in the foreach loop. We suppose

Algorithm 2: *compute_vertex*

Data: A scaffold graph (G^*, M^*), a partial solution S and a vertex v.

1 $[\mathcal{F}(v)] \leftarrow \varnothing$; $[\mathcal{P}(v)] \leftarrow \varnothing$; $[\mathcal{F}(v), 0] \leftarrow \{0\}$;
2 $C \leftarrow \{c_1, \ldots, c_k\}$: list of children linked to v;
3 **foreach** $c_t \in C(v)$ **do**
4 *compute_clique*(c_t);
5 $[\mathcal{F}'] \leftarrow [\mathcal{F}(v)]$;
6 $[\mathcal{P}'] \leftarrow [\mathcal{P}(v)]$;
7 **if** $\exists uv \in E(G^*(v)) \cap S$ **then**
8 **if** $u \in c_t$ **then**
9 $[\mathcal{P}(v)] \leftarrow juxtapose(\{\mathcal{P}'\}, \{\mathcal{P}(c_t)\})$
10 **else**
11 $[\mathcal{P}(v)] \leftarrow juxtapose(\{\mathcal{P}'\}, \{\mathcal{F}(c_t), \mathcal{P}(c_t)\})$
12 **else**
13 $[\mathcal{F}(v)] \leftarrow juxtapose(\{\mathcal{F}'\}, \{\mathcal{F}(c_t), \mathcal{P}(c_t)\})$
14 $[\mathcal{P}(v)] \leftarrow juxtapose(\{\mathcal{P}'\}, \{\mathcal{F}(c_t), \mathcal{P}(c_t)\})$
 $\cup\ juxtapose(\{\mathcal{F}'\}, \{\mathcal{P}(c_t)\})$

that the values of $[\mathcal{F}']$ and $[\mathcal{P}']$ computed by the foreach is loop at the previous step are correct for the graph G_{t-1}. We show that the values of $[\mathcal{F}(v)]$ and $[\mathcal{P}(v)]$ computed in this step are correct for the graph G_t. Suppose also that value of $[\mathcal{F}(c_t)]$ and $[\mathcal{P}(c_t)]$ provided by the function *compute_clique*(c_t) are correct. Let e_t be the edge linking the vertex v with the clique c_t.

- A solution S of G_i is frozen if and only if none of the incident edge to v is in S. Since no such edge belongs to any solution of $G^*(c_t)$, $\mathcal{F}(v)$ is the complete composition of $\mathcal{F}'(v)$ and the set of all solution of $G^*(v)$ with the juxtaposition operation. Thus, the assignment line 7 is correct.
- A solution S of $G^*(v)$ is extensible if and only if S contains exactly one edge incident to v. It is the case if the subsolution of S in G_{t-1} is extensible or if e_t belongs to S. Since e_t can belong to S if the subsolution S in $G^*(c_t)$ is extensible and the subsolution of S is frozen, $\mathcal{P}(v)$ is then the union of (1) the complete composition of $\mathcal{P}'(v)$ and the set of all solution of $G^*(v)$ with the juxtaposition operation and (2) the complete composition of $\mathcal{F}'(v)$ and $\mathcal{P}(c_t)$ with the juxtaposition operation. The addition of e_t extends an alternating path and do not change its number of path. Thus, the assignment line 8 is correct.

Alternating Element. Let c be a clique of G^* and e be an alternating element of c such that e does not contain the upper door of c. We show in this part how to compute the table entries for the sets $\mathcal{C}(e), \mathcal{F}(e)$ and $\mathcal{P}(e)$. If e is a $u - v$-path, then the idea is to merge the computed table entries of u and v and juxtapose the frozen solutions of the inner vertices. If e is an alternating cycle, then there is no choice to do and the only solution containing S is frozen.

Lemma 3. *For any alternating element e, the values of the table entries provided by Algorithm 3 are correct for the sets $\mathcal{C}(e), \mathcal{F}(e)$ and $\mathcal{P}(e)$.*

Algorithm 3: compute_alternating_element

Data: A scaffold graph (G^*, M^*), a partial solution S and an alternating element e with vertices $\{v_0, v_1, \ldots, v_k\}$.

1 **foreach** $v \in p$ **do** *compute_vertex*(v) ;
2 **if** *e is an alternating cycle* **then**
3 $\quad [\mathcal{F}(e)] \leftarrow juxtapose(\{e\}, \{\mathcal{F}(v_0)\}, \ldots, \{\mathcal{F}(v_k\})$;
4 $\quad [\mathcal{C}(e)] \leftarrow \varnothing$;$[\mathcal{A}(e)] \leftarrow \varnothing$; $[\mathcal{P}(e)] \leftarrow \varnothing$;
5 **else**
6 $\quad [\mathcal{I}_e] \leftarrow juxtapose(\{\mathcal{F}(v_1)\}, \ldots, \{\mathcal{F}(v_{k-1}\})$;
7 $\quad [\mathcal{C}(e)] \leftarrow juxtapose(\{e\}, \{\mathcal{F}(v_0)\}, \{\mathcal{F}(v_k)\}, \{\mathcal{I}_e\})$;
8 $\quad [\mathcal{F}(e)] \leftarrow merge_3(\{e\}, \{\mathcal{P}(v_0)\}, \{\mathcal{P}_v(v_k)\}, \{\mathcal{I}_e\})$;
$\quad \cup \; close_1(\{e\}, \{\mathcal{F}(v_0)\}, \{\mathcal{F}(v_k)\}, \{\mathcal{I}_e\})$;
9 $\quad [\mathcal{P}(e)] \leftarrow merge_2(\{e\}, \{\mathcal{P}(u)\}, \{\mathcal{F}(v)\}, \{\mathcal{I}_e\})$;
$\quad \cup \; merge_2(\{e\}, \{\mathcal{F}(u)\}, \{\mathcal{P}(v)\}, \{\mathcal{I}_e\})$;

Note that the only possibility to obtain an absorbent solution of $G^*(e)$ is when e is a path and become closed into an alternating cycle. However, suppose that an absorption operation is done in the function *compute_subclique*. The resulting solution can also be obtained by a closing operation with a solution in $\mathcal{C}(e)$. Thus, to avoid recurrence, the value of $[\mathcal{A}(e)]$ is not provided.

Proof. Suppose that the values of the table entries provided by the function *compute_vertex* are correct. First note that, for each inner vertex v_t of e, the subsolutions of $G^*(v_t)$ are necessarily frozen, then a solution of $G^*(e)$ contains a juxtaposition of frozen solutions of the inner vertices of e. If e is an alternating cycle, then the only possible solution is obtained by the juxtaposition of frozen solutions of the inner vertices and the alternating cycle e. Thus, the assignment line 5 is correct. Suppose that e is a partial path. All possible values of the juxtaposition of the frozen solutions of the inner vertices are assigned in the table entry $[\mathcal{I}_e]$.

- A solution S' of $G^*(e)$ is closeable if and only if the degree of the extremities of e are equal to one. Then, the subsolutions of S' in $G^*(v_0)$ and $G^*(v_k)$ are frozen. Alternatively, any juxtaposition of two frozen solutions of $G^*(v_0)$ and $G^*(v_k)$ and the set $\mathcal{I}_e$ gives a closeable solution of $G^*(e)$. Thus, the assignment line 10 is correct.
- A solution S' of $G^*(e)$ is frozen if and only if the degree of the extremities of e are equal to two. It is the case if (1) the subsolutions of S' in $G^*(v_0)$ and $G^*(v_k)$ are extensible or (2) the subsolutions of S' in $G^*(v_0)$ and $G^*(v_k)$ are frozen and e is closed into an alternating cycle. Alternatively, any extensible solutions of $G^*(v_0)$ and $G^*(v_k)$ are merged by the addition of the edges of S and. It gives a frozen solution of $G^*(e)$. Similarly, the juxtaposition of two frozen solution of $G^*(v_0)$ and $G^*(v_k)$ conjugated with the closing of e gives a frozen solution of $G^*(e)$. Thus, the assignment line 11 is correct.
- A solution S' of $G^*(e)$ is extensible and not closeable if and only if exactly one vertex in $\{v_0, v_k\}$ has degree one. Then, exactly one subsolution of S in $G^*(v_0)$ or $G^*(v_k)$ is extensible. Alternatively, any juxtaposition of exactly

one frozen solution and one extensible solution of $G^*(v_0)$ and $G^*(v_k)$ gives an extensible solution of $G^*(e)$. Thus, the assignment line 12 is correct.

Subclique. Let c' be a subclique of G^*. We show in this part how to compute the table entries for the sets $\mathcal{C}, \mathcal{F}, \mathcal{A}$ and $\mathcal{P}$. The idea is to construct the table entry by merging successively each table entry of the alternating elements of c'. For that, we use at each step an intermediate graph G_i and three intermediate sets $\mathcal{F}_+, \mathcal{A}_+$ and $\mathcal{P}_+$. Let E_i be the i first alternating element of c' and $V_i = \bigcup_{e \in E_i} V(G^*(e))$. G_i is the subgraph of G^* induced by V_i. At step i, a solution $S' \in \mathcal{F}_+$ if and only if (1) S' is a solution of G_i, (2) S' contains a set $C \neq \varnothing$ of closeable paths and (3) $S \setminus C$ is frozen. The sets $\mathcal{A}_+$ and $\mathcal{P}_+$ are defined similarly (only the condition (3) changes).

Lemma 4. *For any subclique c', the value of the table entries provided by Algorithm 4 are correct for the sets $\mathcal{C}(c'), \mathcal{F}(c'), \mathcal{A}(c')$ and $\mathcal{P}(c')$.*

Proof. First, note that if $G^*(c')$ is empty, then the only solution is that containing zero alternating cycle and path and this solution is frozen. Thus, the initial values given in the initialization step (i.e. lines $1-4$) are correct. Now, let e_t be the alternating element considered at step t of the foreach loop. Suppose that the values of $[\mathcal{F}'], [\mathcal{F}_+], [\mathcal{A}'], [\mathcal{A}_+], [\mathcal{P}']$ and $[\mathcal{P}_+]$ computed by the step $t-1$ of the foreach loop are correct for the graph G_{t-1}. We show that the values of the six previous table entries are correct for the next step. Suppose that the values of the $[\mathcal{C}(e_t)], [\mathcal{F}(e_t)]$ and $[\mathcal{P}(e_t)]$, provided by the function *compute_alternating_element* are correct. Let S_1 be a solution of G_{t-1}, S_2 be a solution of $G^*(e_t)$ and S' be a composition of S_1 and S_2.

- If S' is composed by a juxtaposition operation, then S_1 belongs to $\mathcal{F}', \mathcal{A}', \mathcal{P}'$, $\mathcal{F}'_+, \mathcal{A}'_+$ or $\mathcal{P}'_+$ and S_2 belongs to $\mathcal{C}(e_t), \mathcal{F}(e_t)$ or $\mathcal{P}(e_t)$.
- If S' is composed by a merger operation then S_1 belongs to $\mathcal{P}', \mathcal{F}'_+, \mathcal{A}'_+$ $\mathcal{P}'_+$ and S_2 belongs to $\mathcal{C}(e_t)$ or $\mathcal{P}(e_t)$.
- If S' is composed by an absorption operation then S_1 belongs to $\mathcal{A}', \mathcal{A}'_+$ and S_2 belongs to $\mathcal{C}(e_t)$.
- If S' is composed by a closing operation then S_1 belongs to $\mathcal{F}'_+, \mathcal{A}'_+$ or $\mathcal{P}'_+$ and S_2 belongs to $\mathcal{C}'$.

It exists then 31 different complete compositions. If $S_2 \in \mathcal{C}(e_t)$ (resp. $\mathcal{P}(e_t)$ and S' is obtained by a closing operation (resp. merger operation), then S' can be closeable (resp. open), if S_1 contains more than one closeable (resp. extensible) path, or not if S_1 contains a unique closeable (resp. extensible) path. Thus, the complete compositions obtained by a closing operation or a merger operation can belong to two of the six desired sets. However, we can solve this problem and also reduce the number of complete composition to consider by ignoring some composition. Let S_1 be a solution of G_{t-1}, S_2 be a solution of $G^*(e_t)$ and S' be a composition of S_1 and S_2. S' is ignored by the algorithm if a solution S' with the same cardinality can be obtained with another composition.

1. If S' is obtained by a closing operation (resp. merger operation) and S_1 contains more than one closeable (resp. extensible) alternating path. Let p_1 and p_2 be two closeable paths (resp. extensible paths). Let S'_1 be the solution similar to S_1 except that p_1 and p_2 had been closed in an alternating cycle

Algorithm 4: *compute_subclique*

Data: A scaffold graph (G^*, M^*), a partial solution S and a subclique c'.

$[\mathcal{F}(c')] \leftarrow \varnothing$; $[\mathcal{P}(c')] \leftarrow \varnothing$; $[\mathcal{A}(c')] \leftarrow \varnothing$;
$[\mathcal{F}_+] \leftarrow \varnothing$; $[\mathcal{P}_+] \leftarrow \varnothing$; $[\mathcal{A}_+] \leftarrow \varnothing$;
$[\mathcal{F}(c'), 0] \leftarrow \{0\}$;
$E \leftarrow \{e_1, \ldots, e_k\}$: list of alternating elements of c';
foreach $e_t \in E$ **do**
  $compute\_alternating\_element(e_t)$;
  $[\mathcal{F}'] \leftarrow [\overline{\mathcal{F}}(c')]$; $[\mathcal{P}'] \leftarrow [\mathcal{P}(c')]$; $[\mathcal{A}'] \leftarrow [\mathcal{A}(c')]$;
  $[\mathcal{F}'_+] \leftarrow [\mathcal{F}_+]$; $[\mathcal{P}'_+] \leftarrow [\mathcal{P}_+]$; $[\mathcal{A}'_+] \leftarrow [\mathcal{A}_+]$;

  $[\mathcal{F}(c')] \leftarrow juxtapose(\{\mathcal{F}'\}, \{\mathcal{F}(e_t)\})$

  $[\mathcal{F}_+] \leftarrow juxtapose(\{\mathcal{F}'_+\}, \{\mathcal{F}(e_t)\})$
  $\cup\ juxtapose(\{\mathcal{F}', \mathcal{F}_+\}, \{\mathcal{C}(e_t)\})$

  $[\mathcal{A}(c')] \leftarrow juxtapose(\{\mathcal{A}'\}, \{\mathcal{F}(e_t)\})$
  $\cup\ merge_2(\{\mathcal{P}'\}, \{\mathcal{P}(e_t)\})$
  $\cup\ absorb(\{\mathcal{A}'\}, \{\mathcal{C}(e_t)\})$
  $\cup\ close_2(\{\mathcal{F}'_+, \mathcal{A}'_+\}, \{\mathcal{C}(e_t)\})$

  $[\mathcal{A}_+] \leftarrow juxtapose(\{\mathcal{A}, \mathcal{A}'_+\}, \{\mathcal{F}(e_t), \mathcal{C}(e_t)\})$
  $\cup\ merge_2(\{\mathcal{P}'_+\}, \{\mathcal{P}(e_t)\})$
  $\cup\ merge_2(\{\mathcal{F}'_+, \mathcal{A}'_+\}, \{\mathcal{C}(e_t)\})$

  $[\mathcal{P}(c')] \leftarrow juxtapose(\{\mathcal{P}'\}, \{\mathcal{F}(e_t), \mathcal{P}(e_t)\})$
  $\cup\ juxtapose(\{\mathcal{F}', \mathcal{A}'\}, \{\mathcal{P}(e_t)\})$
  $\cup\ merge_2(\{\mathcal{F}'_+\}, \{\mathcal{P}(e_t)\})$
  $\cup\ merge_2(\{\mathcal{P}'\}, \{\mathcal{C}(e_t)\})$
  $\cup\ close_2(\{\mathcal{P}'_+\}, \{\mathcal{C}(e_t)\})$

  $[\mathcal{P}_+] \leftarrow juxtapose(\{\mathcal{P}'_+\}, \{\mathcal{F}(e_t), \mathcal{C}(e_t)\})$
  $\cup\ juxtapose(\{\mathcal{F}'_+, \mathcal{A}'_+, \}, \{\mathcal{P}(e_t)\})$
  $\cup\ juxtapose(\{\mathcal{P}'\}, \{\mathcal{C}(e_t)\})$

end
$[\mathcal{C}(c')] \leftarrow [\mathcal{F}_+] \cup [\mathcal{A}_+] \cup [\mathcal{P}_+]$

(resp. merged in a single alternating path) in a previous step. We can obtain a solution with the same cardinality than S by juxtaposing S'_1 and S_2.

2. If (1) $S_1 \in \mathcal{A}'_+, S_2 \in \mathcal{P}(e_t)$ and S' is obtained by a merger operation, (2) $S_1 \in \mathcal{P}'_+, S_2 \in \mathcal{C}(e_t)$ and S' is obtained by a merger operation or (3) $S_1 \in \mathcal{A}'_+, S_2 \in \mathcal{C}(e_t)$ and S' is obtained by an absorption operation. Let p be the closeable path of S_1 that has be absorbed or merged during the composition. Let S'_1 be the solution similar to S_1 except that all matching edges of p had been absorbed or merged in previous steps. We can obtain a solution with the same cardinality than S by juxtaposing S'_1 and S_2.

Thus, by the first item, after a closing operation, we suppose that the obtained solution does not belong to $\mathcal{F}_+(c'), \mathcal{A}_+(c')$ or $\mathcal{P}_+(c')$. Likewise, after a merging operation between a solution in $\mathcal{P}'$ or $\mathcal{P}'_+$ and a solution in $\mathcal{P}(e_t)$, the obtained solution does not belong to $\mathcal{P}(c')$ or $\mathcal{P}_+(c')$. Then, the complete composition of two sets $\mathcal{S}_1 \in \{\mathcal{F}', \mathcal{A}', \mathcal{P}', \mathcal{F}'_+, \mathcal{A}'_+, \mathcal{P}'_+\}$ and $\mathcal{S}_2 \in \{\mathcal{C}(e_t), \mathcal{F}(e_t), \mathcal{P}(e_t)\}$ is a subset of exactly one of the six desired set. The second item allows us to consider only 28 compositions. Algorithm 4 assigns, for each of this 28 compositions, the value of its table entry to exactly one of the table entry among $[\mathcal{F}(c')], [\mathcal{A}(c')], [\mathcal{P}(c')], [\mathcal{F}_+(c')], [\mathcal{A}_+(c')]$ and $[\mathcal{P}_+(c')]$. The correct assignation of a particular composition is easy to verify and we let the reader check the correctness of each particular assignation. Since each solution of G_t belongs to one the six set and is a composition of one solution of G_{t-1} and $G^*(e_t)$, it suffixes to conclude that the six table entries are correct for the graph G_t.

Finally, after the foreach loop, it remains to compute the value of $[\mathcal{C}]$. The sets containing a closeable paths are exactly $\mathcal{F}_+, \mathcal{A}_+$ and $\mathcal{P}_+$. Thus, it suffixes to make the union of their table entries to obtain $[\mathcal{C}(c')]$. Thus, the assignment line 23 is correct.

Clique. Let c be a clique of G^* and d be the upper door of c. We show in this part how to compute the table entries for the sets $\mathcal{F}(c)$ and $\mathcal{P}(c)$. Note that since the edge between $G^*(c)$ and its parent is a bridge, the sets $\mathcal{C}(c)$ and $\mathcal{A}(c)$ are empty. Let e be the aternating element of c containing the upper door d. The idea is to first compute the table entries for the graph $G^*(e)$ and then merge the obtained table entries to the table entries of the subclique. If e is an alternating path and d is an extremity of e, we replace $\mathcal{P}(e)$ by two intermediate sets $\mathcal{P}_d$ and $\mathcal{P}_{d'}$. Let S' be a solution of $G^*(e)$. $S' \in \mathcal{P}_d$ if and only if $S' \in \mathcal{P}(e)$ and d is an extremity of an alternating path of S'. Likewise, $S' \in \mathcal{P}_{d'}$ if and only if $S' \in \mathcal{P}(e)$ and d is not an extremity of an alternating path of S'. Note that $\mathcal{P}(e) = \mathcal{P}_d \cup \mathcal{P}_{d'}$. In order to compute these two sets, we reuse the value of $\mathcal{I}_e$, computed in *compute_alternating_element*.

Lemma 5. *For any clique c, the values of the table entries provided by Algorithm 5 are correct for the sets $\mathcal{F}(c)$ and $\mathcal{P}(d)$.*

Proof. Suppose e is an alternating path and the upper door d of c is an extremity of a e. First, we compute the table entries for the sets $\mathcal{C}(e), \mathcal{F}(dd'), \mathcal{P}_d$ and $\mathcal{P}_{d'}$. Suppose that the values of the table entries provided by *compute_alternating_element*(p) are correct for the sets $\mathcal{C}(e)$ and $\mathcal{F}(e)$. It remains to compute the table entries for the sets $\mathcal{P}_d$ and $\mathcal{P}_{d'}$. We recall that $\mathcal{I}_e$ is the juxtaposition of all frozen solutions of the inner vertices of e.

- A solution S' of $G^*(e)$ belongs to $\mathcal{P}_d$ if and only if S' belongs to $\mathcal{P}(e)$ and no non-matching edge is incident to d in $G^*(e)$. Thus, $\mathcal{P}_d$ is the juxtaposition of $e, \mathcal{I}_e, \mathcal{F}(d)$ and $\mathcal{P}(d')$. The assignment line 5 is correct.
- Similary, a solution S' of $G^*(e)$ belongs to $\mathcal{P}_{d'}$ if and only if S' belongs to $\mathcal{P}(e)$ and no non-matching edge is incident to d' in $G^*(e)$. Thus, $\mathcal{P}_d$ is the juxtaposition of $e, \mathcal{I}_e, \mathcal{P}(d)$ and $\mathcal{F}(d')$. The assignment line 6 is correct.

Further, we show that the table entries computed for the set $\mathcal{F}(c)$ and $\mathcal{P}(c)$ are correct.

Algorithm 5: *compute_clique*

Data: A scaffold graph (G^*, M^*), a partial solution S and a clique c.

1 $d \leftarrow$ upper door of c; $e \leftarrow$ alternating element of c containing d;
2 *compute_subclique*(c'); *compute_alternating_element*(e);
3 **if** *e is an alternating path* ***and*** *d is an extremity of e* **then**
4 $\quad d' \leftarrow$ other extremity of e;
5 $\quad [\mathcal{P}_d] \leftarrow \mathit{juxtapose}(\{e\}, \{\mathcal{F}(d)\}, \{\mathcal{P}(d')\}, \{\mathcal{I}_e\})$
6 $\quad [\mathcal{P}_{d'}] \leftarrow \mathit{juxtapose}(\{e\}, \{\mathcal{P}(d)\}, \{\mathcal{F}(d')\}, \{\mathcal{I}_e\})$
7
8 $\quad [\mathcal{F}(c)] \leftarrow \mathit{juxtapose}(\{\mathcal{F}(e), \mathcal{P}_{d'}\}, \{\mathcal{F}(c'), \mathcal{C}(c'), \mathcal{A}(c'), \mathcal{P}(c')\})$
$\qquad\qquad \cup \ \mathit{merge}_2(\{\mathcal{C}(e), \mathcal{P}_{d'}\}, \{\mathcal{C}(c'), \mathcal{P}(c')\})$
$\qquad\qquad \cup \ \mathit{absorb}(\{\mathcal{C}(e)\}, \{\mathcal{A}(c')\})$
$\qquad\qquad \cup \ \mathit{close}_2(\{\mathcal{C}(e)\}, \{\mathcal{C}(c')\})$
9
10 $\quad [\mathcal{P}(c)] \leftarrow \mathit{juxtapose}(\{\mathcal{C}(e), \mathcal{P}_d\}, \{\mathcal{F}(c'), \mathcal{C}(c'), \mathcal{A}(c'), \mathcal{P}(c')\})\}$
$\qquad\qquad \cup \ \mathit{merge}_2(\{\mathcal{C}(e)\}, \{\mathcal{C}(c'), \mathcal{P}(c')\})$
11 **end**
12 **else**
13 $\quad [\mathcal{F}(c)] \leftarrow \mathit{juxtapose}(\{\mathcal{C}(c'), \mathcal{F}(c'), \mathcal{A}(c'), \mathcal{P}(c')\}, \{\mathcal{C}(e), \mathcal{F}(e), \mathcal{P}(e)\})$
$\qquad\qquad \cup \ \mathit{merge}_2(\{\mathcal{C}(c'), \mathcal{P}(c')\}, \{\mathcal{C}(e), \mathcal{P}(e)\})$
$\qquad\qquad \cup \ \mathit{absorb}_1(\{\mathcal{A}(c')\}, \{\mathcal{C}(e)\})$
$\qquad\qquad \cup \ \mathit{close}(\{\mathcal{C}(c')\}, \{\mathcal{C}(e)\})$
14 $\quad [\mathcal{P}(c)] \leftarrow \varnothing$
15 $\quad$;
16 **end**

– A solution S' of $G^*(c)$ is frozen if and only if S' contains an edge incident to d. It is the case if the subsolution of S' in $G^*(e)$ belongs to $\mathcal{F}(e)$ or $\mathcal{P}_{d'}$ or if S is obtained by a merger operation, an absorption operation or a closing operation. Thus, the assignment line 8 is correct.
– A solution S' of $G^*(c)$ is extensible if and only if S does not contains an edge incident to d. It is the case if the subsolution of S in $G^*(e)$ belongs to $\mathcal{C}(e)$ or $\mathcal{P}_d$ or if S' is obtained by a merger operation and d' is an extremity of an alternating path in the subsolution of S in $G^*(e)$. Thus, the assignment line 10 is correct.

Now suppose that the upper door d of c is an inner vertex of e. In that case, a subsolution S' of $G^*(c)$ is necessarily frozen. Then any feasible composition of a solution of $G^*(c')$ and a solution of $G^*(e)$ is a frozen solution and thus, the assignment line 13 is correct. Similary, since no extensible solution of $G^*(c)$ exist, the assignment line 14 is correct.

3.1 Feasability function

Finally, we can now provide an answer to the feasibility of finding a solution for Scaffolding problem by using Algorithm 6.

Corollary 1. *Given a partial solution S, Algorithm 6 returns true if and only if (G^*, M^*) can be decomposed into σ_p alternating paths and σ_c alternating cycles. The time complexity of the algorithm is $\mathcal{O}(|V(G^*)| \cdot \sigma_c^2)$.*

Algorithm 6: *Feasibility*

Data: A scaffold graph (G^*, M^*) a partial solution S and two integers σ_p, σ_c

```
root ← root of G* ;
compute_subclique(root);
return σ_p ∈ ([C(root), σ_c] ∪ [F(root), σ_c] ∪ [A(root), σ_c] ∪ [P(root), σ_c])
```

Proof. Since $G^*(root) = G^*$, it exists a solution S with $\sigma_p(S) = \sigma_p$ and $\sigma_c(S) = \sigma_c$, if and only if S belongs to $\mathcal{C}(root) \cup \mathcal{F}(root), \cup\mathcal{A}(root) \cup \mathcal{P}(root)$. Thus the return of the function indicates if such a solution exists and then the algorithm is correct. Concerning the time complexity, the composition operations are in $(O)(\sigma_c^2)$. Thus, without taking in account the reccursive calls, the time complexity of Algorithm 2, Algorithm 3, Algorithm 4 and Algorithm 5 in one iteration of a loop is $\mathcal{O}(\sigma_c^2)$. In Algorithm 2, the number of iterations made by all calls of this function depends on $|CC(G^*)|$ G^* and then the time complexity of all this iterations is $(O)(|CC(G^*)| \cdot \sigma_c^2)$. Similary we can show that the time complexity of the iterations made by all calls of Algorithm 3, Algorithm 4 and Algorithm 5 are $(O)(|V| \cdot \sigma_c^2)$, $(O)(|M^*| \cdot \sigma_c^2)$ and $(O)(|CC(G^*)| \cdot \sigma_c^2)$. Then, the time complexity of all iterations in all function is $(O)((|V)| + |M^*| + |CC(G^*)|) \cdot \sigma_c^2)$ and since the number of matching edges and the number of cliques is bounded by the number of vertices of G^*, we have a time complexity in $\mathcal{O}(|V(G^*)| \cdot \sigma_c^2)$.

4 Approximation Result

4.1 Notations and definitions

The algorithm presented in this section is an adaptation of the one described in [2]. The original algorithm works in complete graph and we adapt it so that it works in connected cluster graph. Let (G^*, M^*, ω) be a scaffold connected cluster graph. The idea of the algorithm is to visit each non matching edge of (G^*, M^*, ω) by decreasing order of weight and chose some of them to be part of the solution S. We start by running Algorithm 6 on (G^*, M^*, ω). This run allows us to both verify if a solution is feasible and to initialize the different table entries. When an edge uv is visited, it can be added in S or removed from (G^*, M^*, ω). When an edge uv is chosen, then all non-matching edges incident to u or v are removed from the list of sorted edges. At each step, we must ensure that we can build a solution with the remaining edges, that is, uv is added in S if and only if we can build σ_p alternating paths and σ_c alternating cycles with the remaining edges.

4.2 Optimization

At each step, we must run the Algorithm 6 in order to check if it possible to construct a solution. However, it is not necessary to update all the table entries at each step. In fact, when an edge uv is tested, it is only necessary to update the table entries of the cliques containing u or v and then, update all the table entries of the ancestors of this cliques.

4.3 Algorithm

Lemma 6. *Algorithm 1 provides a solution for the* (σ_p, σ_c)-SCAFFOLDING *in path connected cluster graph with an approximation ratio of five and a time complexity* $\mathcal{O}(|V| \cdot |E(G^*)| \cdot \sigma_c^2)$. *The approximation ratio is tight.*

Proof. We suppose that the entry of the algorithm is a scaffold graph (G^*, M^*, ω) with non-negative weights and such that G^* is path connected cluster graph. We begin to prove that the algorithm is correct. First, since each time we add an edge e to S, we remove from E all incident non matching edges to e, it is easy to see that the set S induce only paths and cycles.

If it is not possible to build a solution from the graph, then the feasibility condition is not verified and then the algorithm returns an error. Otherwise, since we ensure that the feasibility condition is verified at each step, when the algorithm terminates, then it builds σ_p paths and σ_c cycles.

Now, we prove the approximation ratio. Since they always to any solution, we do not consider the edges of M^* in what follows. Notice that, since there is, for each path, one chosen edge less than the number of involved matching edges, and for a cycle, the same number of chosen edge as the number of involved matching edges, then the number of non-matching edges in every solution is exactly $n-\sigma_p$.

We denote by $e_1, \ldots, e_m$ the edges of the graph, sorted in non-increasing order. We denote by $e_1^A, \ldots, e_{n-\sigma_p}^A$ the edges of the solution S_A given by Algorithm 1, sorted by non-increasing order. In the same way, we denote by $e_1^{opt}, \ldots, e_{n-\sigma_p}^{opt}$ the edges of an optimal solution S_{opt} for the problem, also sorted in non-increasing order. Both sequences $e_1^A, \ldots, e_{n-\sigma_p}^A$ and $e_1^{opt}, \ldots, e_{n-\sigma_p}^{opt}$ are clearly subsequences of $e_1, \ldots, e_m$. For an application $\varphi : S_{opt} \to S_A$, we define both following inequations:

$$\forall e \in S_{opt}, \omega(e) \leq \omega(\varphi(e)) \tag{1}$$

$$\forall e \in S_A, |\varphi^{-1}(\{e\})| \leq 5 \tag{2}$$

Inequation (1) indicates that $\forall e \in E$ in an optimal solution, there exists an edge $\varphi(e) \in S_A$ such that the weight of this latter edge is at least the weight of e. Whereas Inequation (2) states that $\forall e \in S_A$, we may associate e to at most four edges of the optimal solution. In the following, we prove that it is possible to define an application φ satisfying these inequations.

An edge e_i^{opt} in an optimal solution could be not chosen by the algorithm for four main reasons:

- It is eliminated because it belongs to R, when an edge e_j^A is chosen. In this case, we have $\omega(e_j^A) \geq \omega(e_i^{opt})$ because only edges appearing after e_j^A in the ordered list can belong to R. When an edge e_j^A is chosen, it can eliminate at most two edges of optimal solution by updating of the list of edges (see Figure 3). We assign $\varphi(e_i^{opt}) = e_j^A$ in this case. Inequation (1) is satisfied by construction, and Inequation (2) holds when considering only the optimal edges which are eliminated by this way.

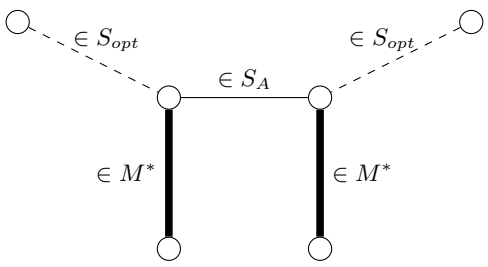

Fig. 3. A greedy edge can eliminated up to two optimal edges by the update_edge function.

- It is eliminated because its addition disconnects the graph and the number of alternating cycles and alternating paths required to cover the graph becomes too big. Such a case can appears in two cases.
  - It closes a cycle. In that case, there exists at least one edge e_j^A in this cycle, and since it has been chosen before the algorithm considers e_i^{opt}, we necessarily have $\omega(e_j^A) \geq \omega(e_i^{opt})$. Thus, we assign $\varphi(e_i^{opt}) = e_j^A$. Inequation (1) is satisfied by construction. The edge e_j^A has been already chosen, may have eliminated at most two optimal edges, but Inequation (2) is still satisfied.
  - It closes a door d and one bridge dd' incident to d is necessary to construct a solution with the remaining edges. There exists a door d'' which has been closed by an edge e_j^A in a previous step and this closing forces dd' to belong to the S_A. Since closing a door increases by at most one the minimum number of alternating paths required to cover the graph, the closing of d'' forces at most one bridge of G^* to belong in S_A. Thus, the closing of d'' prevents d and d' from closing, two edges of S^{opt} can be associated to e_j^A. Inequation (1) is satisfied by construction. The edge e_j^A may have eliminated at most two optimal edges in R and prevent the closing of a cycle, but Inequation (2) is still satisfied.
- It is eliminated because it try to merge two paths p_1 and p_2. If e_i^{opt} is not a bridge and p_1 and p_2 are a single-edge paths, then the number of alternating cycles and paths are reached in S, that is $\sigma_c = c$, $\sigma_p = p$ and $S = S_A$. Then we can find an edge e_j^A such that $|\varphi^-1(e_j^A)| = 0$ and we assign $\varphi(e_i^{opt}) = e_j^A$. Inequations (1) and (2) are satisfied by construction. Otherwise, the algorithm eliminates e_i^{opt} because one of the merged path must be closed into a cycle to reach the correct number of alternating cycles. Otherwise, it exists an edge e_j^A in S_A considered before e_i^{opt} in the algorithm such that $|\varphi^-1(e_j^A)| \leq 3$ (since otherwise the path would be already closed into a cycle) and then we assign $\varphi(e_i^{opt}) = e_j^A$. Inequations (1) and (2) are satisfied by construction.

From the previous discussion and by Inequations (1) and (2), clearly we have:

$$\omega(S_{opt}) \leq \omega(\varphi(S_{opt})) \leq 5 \times \omega(S_A).$$

The ratio is tight, as shown by the example depicted in Figure 4.3.

Concerning the complexity, the sort of the edges can be done in $\mathcal{O}(|V(G^*)|log|E(G^*)|)$. The feasability function is called $|E(G^*)|$ times. Thus, the time complexity of the algorithm is $\mathcal{O}(|E(G^*)| \cdot |V(G^*)| \cdot \sigma_c^2)$.

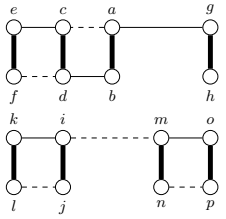

Fig. 4. The approximation ratio of five for the greedy algorithm is tight. The matching edges are bold, the dashed edges belong the approximate solution and the solid edges belong to the optimal solution. G^* is composed by the cliques $C_1 = \{a, b, c, d, e, f\}, C_2 = \{g, h\}, C_3 = \{i, j, k, l\}$ and $C_4 = \{m, n, o, p\}$. All edges are valued by zero except ac and the edges of S_{opt}. We suppose that $\sigma_p = 3$ and $\sigma_c = 0$, and that the greedy algorithm choose first "the wrong edge" ac. Consequently, the solution S_A given by the greedy algorithm is of weight 1, whereas an optimal solution would be of weight 5.

5 Conclusion

We presented in this paper the first polynomial-time algorithm approximating the scaffolding problem on non-complete graphs. Using a dynamic programming approach, we exploited the tree-like nature of connected cluster graph to extend the feasibility function and the analysis of the approximation ratio. A natural extension of this work would be to explore the practical aspects of this algorithm. Since connected cluster graphs aim to model real instances, we intend to measure in what extend this algorithm provides good results on them. We expect the ratio on real instances to be close to one, as for the greedy algorithm on cliques [1]. We may also explore the possibility to exploit randomized algorithms framework to improve this ratio.

References

[1] A. Chateau and R. Giroudeau. Complexity and Polynomial-Time Approximation Algorithms around the Scaffolding Problem. In *Proc. AlCoB '14*, volume 8542 of *LNCS*, pages 47–58. Springer, 2014. ISBN 978-3-319-07952-3.

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

[3] Z. Chen, Y. Harada, E. Machida, F. Guo, and L. Wang. Better approximation algorithms for scaffolding problems. In Daming Zhu and Sergey Bereg, editors, *Frontiers in Algorithmics: 10th International Workshop, FAW 2016, Qingdao, China, June 30- July 2, 2016, Proceedings*, pages 17–28. Springer International Publishing, 2016.

[4] C. Dallard, M. Weller, A. Chateau, and R. Giroudeau. Instance guaranteed ratio on greedy heuristic for genome scaffolding. In *COCOA*, volume 10043 of *Lecture Notes in Computer Science*, pages 294–308. Springer, 2016.

[5] 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.

[6] E. R. Mardis. DNA sequencing technologies: 2006-2016. *Nat Protoc*, 12(2):213–218, Feb 2017.

[7] J. R. Miller, P. Zhou, J. Mudge, J. Gurtowski, H. Lee, T. Ramaraj, B. P. Walenz, J. Liu, R. M. Stupar, R. Denny, L. Song, N. Singh, L. G. Maron, S. R. McCouch, W. R. McCombie, M. C. Schatz, P. Tiffin, N. D. Young, and K. A. T. Silverstein. Hybrid assembly with long and short reads improves discovery of gene family expansions. *BMC Genomics*, 18(1):541, 07 2017.

[8] M. Weller, A. Chateau, and R. Giroudeau. On the complexity of scaffolding problems: from cliques to sparse graphs. In Zaixin Lu, Donghyun Kim, Weili Wu, Wei Li, and Ding-Zhu Du, editors, *Combinatorial Optimization and Applications: 9th International Conference, COCOA 2015, Houston, TX, USA, December 18-20, 2015, Proceedings*, pages 409–423. Springer International Publishing, 2015.

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

[10] 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. doi: 10.1007/s00453-018-0405-x. URL https://doi.org/10.1007/s00453-018-0405-x.

A Appendix

A.1 Algorithms

Algorithm 7: *juxtapose*

Data: $\mathcal{S}^1 = \{\mathcal{S}_1^1, \mathcal{S}_2^1, \dots\}, \dots, \mathcal{S}^k = \{\mathcal{S}_1^k, \mathcal{S}_2^k, \dots\}$: sets of sets of solutions.

if $k = 0$ **then**
  $[\mathcal{S}] \leftarrow 0$;
end
$[\mathcal{I}] \leftarrow juxtapose(S^2, \dots, S^k)$;
forall $i \in [0, \sigma_c]$ **do**
  forall $j \in [0, \sigma_c - i]$ **do**
    $[\mathcal{S}, i+j] \leftarrow [\mathcal{S}, i] + \bigcup_{S \in \mathcal{S}^1} [S, j]$
  end
end
return $[\mathcal{S}]$

Algorithm 8: *merge$_t$ or absorb*

Data: $\mathcal{S}^1 = \{\mathcal{S}_1^1, \mathcal{S}_2^1, \dots\}, \dots, \mathcal{S}^k = \{\mathcal{S}_1^k, \mathcal{S}_2^k, \dots\}$: sets of sets of solutions,
t: number of paths to merge ($t = 2$ in the absorb function).

forall $i \in [0, \sigma_c]$ **do**
  forall $j \in [0, \sigma_c - i]$ **do**
    $[\mathcal{S}, i+j] \leftarrow \bigcup_{S \in \mathcal{S}^1} [S, i] + \bigcup_{S' \in \mathcal{S}^2} [S', j] + \{-(t-1)\}$
  end
end
if $k \neq 2$ **then**
  $[\mathcal{S}] \leftarrow juxtapose(\{S\}, S^3, \dots, S^k)$;
end
return $[\mathcal{S}]$

Algorithm 9: *close$_t$*

Data: $\mathcal{S}^1 = \{\mathcal{S}_1^1, \mathcal{S}_2^1, \dots\}, \dots, \mathcal{S}^k = \{\mathcal{S}_1^k, \mathcal{S}_2^k, \dots\}$: sets of sets of solutions,
t: number of paths to close.

forall $i \in [0, \sigma_c]$ **do**
  forall $j \in [0, \sigma_c - i]$ **do**
    $[\mathcal{S}, i+j+1] \leftarrow \bigcup_{S \in \mathcal{S}^1} [S, i] + \bigcup_{S' \in \mathcal{S}^2} [S', j] + \{-t\}$
  end
end
if $k \neq 2$ **then**
  $[\mathcal{S}] \leftarrow juxtapose(\{S\}, S^3, \dots, S^k)$;
end
return $[\mathcal{S}]$