An approximation algorithm for maximum internal spanning tree

Zhi-Zhong Chen1 · Youta Harada1 · Fei Guo2 · Lusheng Wang3

Published online: 15 January 2018
© Springer Science+Business Media, LLC, part of Springer Nature 2018

Abstract Given a graph $G$, the maximum internal spanning tree problem (MIST for short) asks for computing a spanning tree $T$ of $G$ such that the number of internal vertices in $T$ is maximized. MIST has possible applications in the design of cost-efficient communication networks and water supply networks and hence has been extensively studied in the literature. MIST is NP-hard and hence a number of polynomial-time approximation algorithms have been designed for MIST in the literature. The previously best polynomial-time approximation algorithm for MIST achieves a ratio of $\frac{3}{4}$. In this paper, we first design a simpler algorithm that achieves the same ratio and the same time complexity as the previous best. We then refine the algorithm into a new approximation algorithm that achieves a better ratio (namely, $\frac{13}{17}$) with the same time complexity. Our new algorithm explores much deeper structure of the problem than the previous best. The discovered structure may be used to design even better approximation or parameterized algorithms for the problem in the future.
Keywords Approximation algorithms · Graph algorithms · Spanning trees · Path-cycle covers

1 Introduction

The maximum internal spanning tree problem (MIST for short) requires the computation of a spanning tree $T$ in a given graph $G$ such that the number of internal vertices in $T$ is maximized. MIST has possible applications in the design of cost-efficient communication networks (Salamon and Wiener 2008) and water supply networks (Binkele-Raible et al. 2013). Unfortunately, MIST is clearly NP-hard because the problem of finding a Hamiltonian path in a given graph is NP-hard (Garey and Johnson 1979) and can be easily reduced to MIST. MIST is in fact APX-hard (Li and Zhu 2014) and hence does not admit a polynomial-time approximation scheme.

Since MIST is APX-hard, it is of interest to design polynomial-time approximation algorithms for it that achieve a constant ratio as close to 1 as possible. Indeed, Prieto and Sliper (2003) presented a polynomial-time approximation algorithm for MIST achieving a ratio of $\frac{1}{2}$. Their algorithm is based on local search. By slightly modifying Prieto and Sliper’s algorithm, Salamon and Wiener (2008) then obtained a faster (linear-time) approximation algorithm achieving the same ratio. Salamon and Wiener (2008) also considered two special cases of MIST. More specifically, they designed a polynomial-time approximation algorithm for the special case of MIST restricted to claw-free graphs that achieves a ratio of $\frac{2}{3}$, and also designed a polynomial-time approximation algorithm for the special case of MIST restricted to cubic graphs that achieves a ratio of $\frac{5}{6}$. Salamon (2009) later proved that the approximation algorithm by Salamon and Wiener (2008) indeed achieves a performance ratio of $\frac{3}{r+1}$ for the special case of MIST restricted to $r$-regular graphs. Based on local optimization, Salamon (2009a) further came up with an $O(n^4)$-time approximation algorithm for the special of MIST restricted to graphs without leaves that achieves a ratio of $\frac{4}{7}$. The algorithm by Salamon (2009a) was subsequently simplified and re-analyzed by Knauer and Spoerhase (2009) so that it runs faster (in cubic time) and achieves a better ratio (namely, $\frac{3}{5}$) for (the general) MIST. Li et al. (2014) even went further by showing that a deeper local search than those by Knauer and Spoerhase (2009) and Salamon (2009a) can achieve a ratio of $\frac{7}{5}$ for MIST. Recently, Li and Zhu (2014) came up with a polynomial-time approximation algorithm for MIST that achieves a ratio of $\frac{3}{4}$. Unlike the other previously known approximation algorithms for MIST, the algorithm by Li and Zhu (2014) is based on a simple but crucial observation that the maximum number of internal vertices in a spanning tree of a graph $G$ can be bounded from above by the maximum number of edges in a triangle-free path-cycle cover of $G$.

In the weighted version of MIST (WMIST for short), each vertex of the given graph $G$ has a nonnegative weight and the objective is to find a spanning tree $T$ of $G$ such that the total weight of internal vertices in $T$ is maximized. Salamon (2009a) designed an $O(n^4)$-time approximation for WMIST that achieves a ratio of $\frac{1}{2\Delta-3}$, where $\Delta$ is the maximum degree of a vertex in the input graph. Salamon (2009a) also considered the special case of WMIST restricted to claw-free graphs without leaves, and designed an $O(n^4)$-time approximation algorithm for the special case that achieves...
a ratio of $\frac{1}{3}$. Subsequently, Knauer and Spoerhase (2009) proposed a polynomial-time approximation algorithm for (the general) WMIST that achieves a ratio of $\frac{1}{3} - \epsilon$ for any constant $\epsilon > 0$.

In the parameterized version of MIST (PMIST for short), we are asked to decide whether a given graph $G$ has a spanning tree with at least a given number $k$ of internal vertices. PMIST and its special cases and variants have also been extensively studied in the literature (Binkele-Raible et al. 2013; Coben et al. 2010; Fomin et al. 2012, 2013; Li et al. 2015; Prieto and Sliper 2003; Prieto 2005; Prieto and Sloper 2005). The best known kernel for PMIST is of size $2k$ and it leads to the fastest known algorithm for PMIST with running time $O(4^k n^{O(1)})$ (Li et al. 2015).

In this paper, we first give a new approximation algorithm for MIST that is simpler than the one by Li and Zhu (2014) but achieves the same approximation ratio and time complexity. In more details, the time complexity is dominated by that of computing a maximum triangle-free path-cycle cover in a graph. We then show that the algorithm can be refined into a new approximation algorithm for MIST that has the same time complexity as the algorithm by Li and Zhu (2014) but achieves a better ratio (namely, $\frac{13}{17}$). To obtain our algorithm, we use three new main ideas. The first main idea is to bound the maximum number of internal vertices in a spanning tree of a graph $G$ by the maximum number of edges in a special (rather than general) triangle-free path-cycle cover of $G$. Roughly speaking, we can figure out that certain vertices in $G$ must be leaves in an optimal spanning tree of $G$, and hence we can require that the degrees of these vertices be at most 1 when computing a maximum triangle-free path-cycle cover $C$ of $G$. In this sense, $C$ is special and can have significantly fewer edges than a maximum (general) triangle-free path-cycle cover of $G$, and hence gives us a tighter upper bound. The second idea is to carefully modify $C$ into a spanning tree $T$ by local improvement. Unfortunately, we can not always guarantee that the number of internal vertices in $T$ is at least $\frac{13}{17}$ times the number of edges in $C$. Our third idea is to show that if this unfortunate case occurs, then an optimal spanning tree of $G$ cannot have so many internal vertices. Although the improvement of approximation ratio may not look so significant, our ideas may be used to design even better approximation or parameterized algorithms for MIST and related problems in the future. Indeed, we have applied some of the ideas to obtain a $\frac{1}{2}$-approximation algorithm for WMIST (Chen et al. 2016).

The remainder of this paper is organized as follows. Section 2 gives basic definitions that will be used in the remainder of the paper. Section 3 presents a simple approximation algorithm for MIST that achieves a ratio of $\frac{3}{4}$. The subsequent sections are devoted to refining the algorithm so that it achieves a better ratio.

2 Basic definitions

Throughout this chapter, a graph means a simple undirected graph (i.e., it has neither parallel edges nor self-loops).

Let $G$ be a graph. We denote the vertex set of $G$ by $V(G)$, and denote the edge set of $G$ by $E(G)$. For a subset $U$ of $V(G)$, $G - U$ denotes the graph obtained from $G$ by removing the vertices in $U$ (together with the edges incident to them), while $G[U]$. 
denotes $G - (V(G) \setminus U)$. We call $G[U]$ the subgraph of $G$ induced by $U$. For a subset $F$ of $E(G)$, $G - F$ denotes the graph obtained from $G$ by removing the edges in $F$. An edge $e$ of $G$ is a bridge of $G$ if $G - \{e\}$ has more connected components than $G$, and is a non-bridge otherwise. A vertex $v$ of $G$ is a cut-point if $G - \{v\}$ has more connected components than $G$.

Let $v$ be a vertex of $G$. The neighborhood of $v$ in $G$, denoted by $N_G(v)$, is $\{u \mid \{v, u\} \in E(G)\}$. The degree of $v$ in $G$, denoted by $d_G(v)$, is $|N_G(v)|$. If $d_G(v) = 0$, then $v$ is an isolated vertex of $G$. If $d_G(v) \leq 1$, then $v$ is a leaf of $G$; otherwise, $v$ is a non-leaf of $G$. We use $L(G)$ to denote the set of leaves in $G$.

Let $H$ be a subgraph of $G$. $N_G(H)$ denotes $\bigcup_{v \in V(H)} N_G(v) \setminus V(H)$. A port of $H$ is a $u \in V(H)$ with $N_G(u) \setminus V(H) \neq \emptyset$. When $H$ is a path, $H$ is dead if neither endpoint of $H$ is a port of $H$, while $H$ is alive otherwise. $H$ and another subgraph $H'$ of $G$ are adjacent in $G$ if $V(H) \cap V(H') = \emptyset$ but $N_G(H) \cap V(H') \neq \emptyset$ (or equivalently, $N_G(H') \cap V(H) \neq \emptyset$).

A cycle in $G$ is a connected subgraph of $G$ in which each vertex is of degree 2. A path in $G$ is either a single vertex of $G$ or a connected subgraph of $G$ in which exactly two vertices are of degree 1 and the others are of degree 2. A vertex $v$ of a path $P$ in $G$ is an endpoint of $P$ if $d_P(v) \leq 1$, and is an internal vertex of $P$ if $d_P(v) = 2$. The length of a cycle or path $C$ is the number of edges in $C$ and is denoted by $|C|$. A $k$-cycle is a cycle of length $k$, while a $k$-path is a path of length $k$. A tree (respectively, cycle) component of $G$ is a connected component of $G$ that is a tree (respectively, cycle). In particular, if a tree component $T$ of $G$ is indeed a path (respectively, $k$-path), then we call $T$ a path (respectively, $k$-path) component of $G$.

A tree-cycle cover (TCC for short) of $G$ is a subgraph $H$ of $G$ such that $V(H) = V(G)$ and each connected component of $H$ is a tree or cycle. Let $H$ be a TCC of $G$. $H$ is a Hamiltonian path (respectively, cycle) of $G$ if $H$ is a path (respectively, cycle), and is a spanning tree of $G$ if $H$ is a tree. $H$ is a path-cycle cover (PCC for short) of $G$ if each tree component of $H$ is a path. $H$ is a path cover of $G$ if $H$ has only path components. A triangle-free TCC (TFTCC for short) of $G$ is a TCC without 3-cycles. Similarly, a triangle-free PCC (TFPCC for short) of $G$ is a PCC without 3-cycles. A TFPCC of $G$ is maximum if its number of edges is maximized over all TFPCCs of $G$. For convenience, let $t(n, m)$ denote the time complexity of computing a maximum TFPCC in a graph with $n$ vertices and $m$ edges. It is known that $t(n, m) = O(n^2 m^2)$ (Hartvigsen 1984).

Suppose that $G$ is connected. The weight of a spanning tree $T$ of $G$, denoted by $w(T)$, is the number of non-leaves in $T$. We use $\text{opt}(G)$ to denote the maximum weight of a spanning tree of $G$. An optimal spanning tree (OST for short) of $G$ is a spanning tree $T$ of $G$ with $w(T) = \text{opt}(G)$.

## 3 A simple 0.75-approximation algorithm

Throughout the remainder of this paper, $G$ means a connected graph for which we want to find an OST. Moreover, $T$ denotes an OST of $G$. For convenience, let $n = |V(G)|$ and $m = |E(G)|$. 

© Springer
3.1 Reduction rules

We want to make $G$ smaller (say, by deleting one or more vertices or edges from $G$) without decreasing $opt(G)$. For this purpose, we define two strongly safe operations on $G$ below. Here, an operation on $G$ is strongly safe if performing it on $G$ does not change $opt(G)$ (Figs. 1, 2, 3, 4, 5, 6).

**Operation 1** If $|V(G)| > 3$ and $E(G)$ contains two edges $\{u_1, v\}$ and $\{u_2, v\}$ such that both $u_1$ and $u_2$ are leaves of $G$, then delete $u_2$.

**Operation 2** If for a non-bridge $e = \{u_1, u_2\}$ of $G$, $G - \{u_i\}$ has a connected component $K_i$ with $u_{3-i} \notin V(K_i)$ for each $i \in \{1, 2\}$, then delete $e$. (Comment: When $|V(K_1)| = |V(K_2)| = 1$, Li and Zhu (2014) showed that Operation 2 is strongly safe.)

**Lemma 1** (Li and Zhu 2014) Operation 1 is strongly safe.
Fig. 4 Operations 8 through 11, where the wavy line is a path and each dotted edge will be deleted

Fig. 5 Operations 12 through 14, where each wavy line or curve is a path, each dotted edge will be deleted, and each bold edge will be added

Fig. 6 Operations 15 through 23, where the filled circle is a port, each wavy line or curve is a path, each filled triangle is a tree, each dotted edge will be deleted, and each bold edge will be added
Lemma 2 Operation 2 is strongly safe.

Proof If \( e \notin E(T) \), we are done. So, assume that \( e \in E(T) \). Obviously, at least one vertex \( v_1 \) of \( K_e \) is adjacent to \( u_1 \) in \( T \) because \( T \) is connected. So, \( \{u_2, v_1\} \subseteq N_T(u_1) \).

Similarly, \( \{u_1, v_2\} \subseteq N_T(u_2) \) for some vertex \( v_2 \) of \( K_2 \). Moreover, since \( e \) is a non-bridge of \( G \), \( G - \{u_1, u_2\} \) has a connected component \( K_3 \) (other than \( K_1 \) and \( K_2 \) with \( \{u_1, u_2\} \subseteq N_G(K_3) \). Since \( T \) is connected, \( u_1 \) or \( u_2 \) is adjacent to a vertex \( v_3 \) of \( K_3 \) in \( T \). We assume that \( v_3 \in N_T(u_1) \); the other case is similar. Then, after deleting \( e \) from \( T \), only \( u_2 \) may become a new leaf. If \( u_2 \) becomes a leaf in \( T \) – \( e \), then all vertices of \( K_3 \) must belong to the component tree of \( T \) – \( e \) containing \( u_1 \) and hence adding an arbitrary edge \( \{u_2, v_4\} \) of \( G \) with \( v_4 \in V(K_3) \) to \( T \) – \( e \) yields a new OST of \( G \). So, we may assume that \( u_2 \) does not become a leaf in \( T \) – \( e \). Then, since \( e \) is a non-bridge of \( G \), \( G \) must have an edge \( \{x_1, x_2\} \) such that for each \( i \in \{1, 2, 3\} \), \( x_i \) belongs to the component tree of \( T \) – \( e \) containing \( u_i \). Now, adding the edge \( \{x_1, x_2\} \) to \( T \) – \( e \) yields a new OST of \( G \). \( \square \)

An operation on \( G \) is weakly safe if performing it on \( G \) yields one or more graphs \( G_1, \ldots, G_k \) such that (1) \( |V(G)| \geq \sum_{i=1}^{k} |V(G_i)|, |E(G)| \geq \sum_{i=1}^{k} |E(G_i)|, \) and \( |V(G)|+|E(G)| > \sum_{i=1}^{k} |V(G_i)|+\sum_{i=1}^{k} |E(G_i)| \), (2) \( opt(G) \leq \sum_{i=1}^{k} opt(G_i) + c \) for some nonnegative integer \( c \), and (3) given a spanning tree \( T_i \) for each \( G_i \), a spanning tree \( T \) of \( G \) with \( w(T) \geq \sum_{i=1}^{k} w(T_i) + c \) can be computed in linear time. Note that the last two conditions in the definition imply that \( opt(G) = \sum_{i=1}^{k} opt(G_i) + c \).

Operation 3 If \( G \) has a bridge \( e = \{u_1, u_2\} \) such that for each \( i \in \{1, 2\}, u_i \) is a cut-point in the connected component \( G_i \) of \( G - e \) with \( u_i \in V(G_i) \), then obtain \( G_1 \) and \( G_2 \) as the connected components of \( G - e \).

Operation 4 If \( G \) has a cut-point \( v \) such that one connected component \( K \) of \( G - \{v\} \) has at least two but at most 8 vertices, then obtain \( G_1 \) from \( G - V(K) \) by adding a new vertex \( u \) and a new edge \( \{v, u\} \).

The number 8 in the definition of Operation 4 is not essential. It can be chosen at one’s discretion as long as it is a constant. We here choose the number 8, because it will be the smallest number for the proofs of several lemmas in this paper to go through.

Lemma 3 Operation 3 is weakly safe.

Proof First, we want to show that \( opt(G) \leq opt(G_1) + opt(G_2) \). Consider an \( i \in \{1, 2\} \). Since \( u_i \) is a cut-point in \( G_i \), \( d_T(u_i) \geq 3 \). Thus, the degree of \( u_i \) in \( T \) – \( e \) is at least 2. So, one component tree of \( T \) – \( e \) is a spanning tree of \( G_1 \), the other is a spanning tree of \( G_2 \), and their total weights equals \( w(T) \). Thus, \( opt(G) \leq opt(G_1) + opt(G_2) \).

Next, suppose that for each \( i \in \{1, 2\} \), \( T_i \) is a spanning tree of \( G_i \). Since \( u_i \) is a cut-point in \( G_i \), \( d_{T_i}(u_i) \geq 2 \). So, using \( e \) to connect \( T_1 \) and \( T_2 \) into a single tree yields a spanning tree of \( G \) whose weight is \( w(T_1) + w(T_2) \). \( \square \)

Lemma 4 Operation 4 is weakly safe.
Proof Let $K'$ be the graph obtained from $G[V(K) \cup \{v\}]$ by adding a new vertex $u'$ and a new edge $\{u',v\}$. Let $c = \text{opt}(K') - 1$.

First, we want to show that $\text{opt}(G) \leq \text{opt}(G_1) + c$. Since $v$ is a cut-point of $G$, $d_T(v) \geq 2$. Let $T_1$ be the spanning tree of $G_1$ obtained from $T - V(K)$ by adding $u$ and the edge $\{u,v\}$. Further let $T'$ be the spanning tree of $K'$ obtained from $T[\{v\} \cup V(K)]$ by adding $u'$ and edge $\{u',v\}$. Clearly, $w(T) = w(T_1) + w(T') - 1 \leq \text{opt}(G_1) + c$. Thus, $\text{opt}(G) \leq \text{opt}(G_1) + c$.

Next, suppose that $T_1$ is a spanning tree of $G_1$. Let $T'$ be an OST of $K'$. We can obtain a spanning tree $\tilde{T}$ of $G$ from $T_1$ by first deleting $u$, next adding $T'[V(K)]$, and further adding new edges to connect $v$ to those vertices of $V(K)$ that are adjacent to $v$ in $T'$. Obviously, $u \in L(T_1), u' \in L(T'), v \notin L(T_1), v \notin L(T')$, the degree of each vertex $x$ of $T_1$ other than $v$ and $u$ in $\tilde{T}$ is $d_{T_1}(x)$, and the degree of each vertex $y$ of $T'$ other than $v$ and $u'$ in $\tilde{T}$ is $d_{T'}(y)$. Thus, $w(\tilde{T}) = w(T_1) + w(T') - 1 = w(T_1) + c$. □

An operation on $G$ is safe if it is strongly or weakly safe on $G$.

3.2 The algorithm

As in Li and Zhu (2014), the algorithm is based on a lemma which says that $G$ has a path cover $\mathcal{P}$ such that $\text{opt}(G)$ is bounded from above by the number of edges in $\mathcal{P}$. We next state the lemma in a stronger form and give an extremely simple proof.

Lemma 5 Given a spanning tree $\tilde{T}$ of $G$, we can construct a path cover $\mathcal{P}$ of $G$ such that $|E(\mathcal{P})| \geq w(\tilde{T})$ and $d_{\mathcal{P}}(v) \leq 1$ for each leaf $v$ of $\tilde{T}$.

Proof We simply construct $\mathcal{P}$ from $\tilde{T}$ by first rooting $\tilde{T}$ at an arbitrary non-leaf and then for each non-leaf $u$ of $\tilde{T}$, deleting all but one edge between $u$ and its children. □

Now, the outline of the algorithm is as follows.

1. Whenever there is an $i \in \{1,2\}$ such that Operation $i$ can be performed on $G$, then perform Operation $i$ on $G$.
2. Whenever there is an $i \in \{3,4\}$ such that Operation $i$ can be performed on $G$, then perform the following steps:
   (a) Perform Operation $i$ on $G$. Let $G_1, \ldots, G_k$ be the resulting graphs.
   (b) For each $j \in \{1, \ldots, k\}$, compute a spanning tree $T_j$ of $G_j$ recursively.
   (c) Combine $T_1, \ldots, T_k$ into a spanning tree $\tilde{T}$ of $G$ such that $w(\tilde{T}) \geq \sum_{i=1}^{k} w(T_i) + c$.
   (d) Return $\tilde{T}$.
3. If $|V(G)| \leq 8$, then compute and return an OST of $G$ in $O(1)$ time.
4. Compute a maximum TFPCC $\mathcal{C}$ of $G$. (Comment: By Lemma 5, opt$(G) \leq |E(\mathcal{C})|$).
5. Perform a preprocessing on $\mathcal{C}$ without decreasing $|E(\mathcal{C})|$.
6. Transform $\mathcal{C}$ into a spanning tree $\tilde{T}$ of $G$ such that $w(\tilde{T}) \geq \frac{3}{4}|E(\mathcal{C})|$.
7. Return $\tilde{T}$.

Only Steps 5 and 6 are unclear. So, we detail them below. First, Step 5 is done by performing the next three operations until none of them is applicable.
**Operation 5** If $C$ has a dead path component $P$ such that $2 \leq |P| \leq 4$ and $G[V(P)]$ has an alive Hamiltonian path $Q$, then replace $P$ by $Q$.

**Operation 6** If an endpoint $u$ of a path component $P$ of $C$ is adjacent to a vertex $v$ of a cycle $C$ of $C$ in $G$, then combine $P$ and $C$ into a single path by replacing one edge incident to $v$ in $C$ with the edge $\{u, v\}$.

**Operation 7** If an endpoint $u_1$ of a path component $P_1$ of $C$ is adjacent to an internal vertex $u_2$ of another path component $P_2$ in $G$ such that one edge $e'$ incident to $u_2$ in $P_2$ satisfies that combining $P_1$ and $P_2$ by replacing $e'$ with the edge $\{u_1, u_2\}$ yields two paths $Q_1$ and $Q_2$ with $\max(|Q_1|, |Q_2|) > \max(|P_1|, |P_2|)$, then replace $P_1$ and $P_2$ by $Q_1$ and $Q_2$. (Comment For each $i \in \{5, 6, 7\}$, Operation $i$ does not change the maximality of $C$. So, due to the maximality of $C$, no endpoint of a path component $P_1$ of $C$ is adjacent to an endpoint of another path component $P_2$ in $G$.)

**Lemma 6** Immediately after Step 5, the following statements hold:

1. $C$ is a maximum TFPCC of $G$ and hence has at least $\text{opt}(G)$ edges.
2. If a path component $P$ of $C$ is of length at most 3, then $P$ is alive.
3. If an endpoint $v$ of a path component $P$ of $C$ is a port of $P$, then each vertex in $N_G(v) \setminus V(P)$ is an internal vertex of a path component $Q$ of $C$ with $|Q| \geq 2|P| + 2$.

**Proof** We prove the statements separately as follows.

**Statement 1** Immediately before Step 5, $C$ has a maximum TFPCC of $G$. Since Operations 5 through 7 keep $C$ being a TFPCC without changing the number of edges in $C$, Statement 1 holds.

**Statement 2** Let $P$ be a path component of $C$ with $|P| \leq 3$. If $|P| \leq 1$, then $P$ is alive because otherwise $G$ would be disconnected. So, $|P| = 2$ or 3. Let $u_1$ and $u_2$ be the endpoints of $P$. For a contradiction, assume that $P$ is dead. Then, since $G$ is connected, $P$ has at least one internal vertex $x$ adjacent to a vertex $x' \in V(G) \setminus V(P)$ in $G$. If $\{u_1, u_2\} \notin E(G)$, then $G[V(P)]$ has a Hamiltonian path $Q$ in which $x$ is an endpoint, contradicting the fact that Operation 5 cannot be performed on $C$. So, we assume that $\{u_1, u_2\} \notin E(G)$. Now, if $|P| = 2$, then Operation 1 can be performed on $G$, a contradiction. Thus, we further assume that $|P| = 3$. Then, since Operation 4 cannot be performed on $G$, the other internal vertex $y$ (than $x$) of $P$ is adjacent to a vertex $y' \in V(G) \setminus V(P)$ in $G$. Now, if $G[V(P)]$ is not $P$ itself, then Operation 5 can be performed on $C$, a contradiction; otherwise, Operation 2 or 3 can be performed on $G$, a contradiction. Note that it does not matter whether $x' = y'$ or not.

**Statement 3** Suppose that an endpoint $v$ of a path component $P$ of $C$ is a port. Consider an arbitrary $u \in N_G(v) \setminus V(P)$. Since Operation 6 is not applicable on $C$, $u$ appears in a path component $Q$ of $C$. Then, by the comment on Operation 7, $u$ is an internal vertex of $Q$. Let $u_1$ and $u_2$ be the endpoints of $Q$. For each $i \in \{1, 2\}$, let $Q_i$ be the path from $u$ to $u_i$ in $P$. Then, $|Q| = |Q_1| + |Q_2|$. Moreover, since Operation 7 cannot be applied on $C$, $|P| + |Q_i| + 1 \leq |Q|$ for each $i \in \{1, 2\}$. Thus, $2|P| + 2 \leq |Q|$.

We next detail Step 6. First, for each path component $P$ of $C$ with $1 \leq |P| \leq 3$, we select one edge $e_P \in E(G)$ connecting an endpoint of $P$ to a vertex not in $P$, and add
$e_P$ to an initially empty set $M$. Such $e_P$ exists by Statement 2 in Lemma 6. Moreover, by Statement 3 in Lemma 6, the endpoint of $e_P$ not in $P$ appears in a path component $Q$ of $C$ with $|Q| \geq 4$. So, for two path components $P_1$ and $P_2$ in $C$, $e_{P_1} \neq e_{P_2}$. Consider the graph $H$ obtained from $C$ by adding the edges in $M$. Each connected component of $H$ is a cycle of length at least 4 or a tree. Suppose that we modify $H$ by performing the following three steps in turn:

- Whenever $H$ has two cycles $C_1$ and $C_2$ such that some edge $e = \{u_1, u_2\} \in E(G)$ satisfies $u_1 \in V(C_1)$ and $u_2 \in V(C_2)$, delete one edge of $C_1$ incident to $u_1$ from $H$, delete one edge of $C_2$ incident to $u_2$ from $H$, and add $e$ to $H$.

- Whenever $H$ has a cycle $C$, choose an edge $e = \{u, v\} \in E(G)$ with $u \in V(C)$ and $v \notin V(C)$, delete one edge of $C$ incident to $u$ from $H$, and add $e$ to $H$.

- Whenever $H$ has two connected components $C_1$ and $C_2$ such that some edge $e = \{u_1, u_2\} \in E(G)$ satisfies $u_1 \in V(C_1)$ and $u_2 \in V(C_2)$, add $e$ to $H$.

Step 6 is done by obtaining $\tilde{T}$ as the final modified $H$. Obviously, for each cycle $C$ of $C$, at least $|C| - 1 \geq \frac{3}{4}|C|$ vertices of $C$ are internal vertices of $\tilde{T}$. Moreover, for each path component $P$ of $C$ with $|P| \geq 4$, at least $|P| - 1 \geq \frac{3}{4}|P|$ vertices of $P$ are internal vertices of $\tilde{T}$. Furthermore, for each path component $P$ of $C$ with $1 \leq |P| \leq 3$, at least $|P|$ vertices of $P$ are internal vertices of $\tilde{T}$. So, $\tilde{T}$ has at least $\frac{3}{4}|E(C)|$ internal vertices. Obviously, all steps of the algorithm excluding Steps 2b and 4 can be done in $O(|E(G)|^2)$ time. Now, we have the following theorem:

**Theorem 1** The algorithm achieves an approximation ratio of $\frac{3}{4}$ and runs in $O(m^2) + t(n, m)$ time.

In the sequel, we consider how to improve the algorithm. The first idea is to introduce more safe reduction rules (cf. Sect. 4). The second idea is to compute a better upper bound on $\text{opt}(G)$ than that given by a maximum TFPCC (cf. Sect. 5). The third idea is to perform a more sophisticated preprocessing on $C$ (cf. Sect. 6). The last idea is to transform $C$ into a spanning tree of $G$ more carefully (cf. Sect. 7).

## 4 More safe reduction rules

In addition to the four safe reduction rules in Sect. 3.1, we further introduce the following rules.

**Operation 8** If for four vertices $u_1, \ldots, u_4$, $N_G(u_3) = N_G(u_4) = \{u_1, u_2\}$, $G - \{u_2\}$ has a connected component $K$ with $u_1 \notin V(K)$, then delete the edge $e = \{u_2, u_3\}$.

**Operation 9** If for five vertices $u_1, \ldots, u_5$, $N_G(u_3) = N_G(u_4) = N_G(u_5) = \{u_1, u_2\}$, then delete the edge $e = \{u_2, u_3\}$.

**Operation 10** If for two vertices $u$ and $v$ of $G$, $G - \{u, v\}$ has a connected component $K$ with $|V(K)| \leq 6$ such that $V(G) \neq V(K) \cup \{u, v\}$ and $G[V(K) \cup \{u, v\}]$ has a Hamiltonian path $P$ from $u$ to $v$, then delete all edges of $G[V(K) \cup \{u, v\}]$ that do not appear in $P$. 
Operation 11 If $G$ has an edge $e = \{u_1, u_2\}$ with $d_G(u_1) = d_G(u_2) = 2$, then obtain $G_1$ from $G$ by merging $u_1$ and $u_2$ into a single vertex $u_1u_2$.

Lemma 7 Operation 8 is strongly safe.

Proof If $e \notin E(T)$, we are done. So, assume that $e \in E(T)$. Obviously, at least one vertex $v$ of $K$ is adjacent to $u_2$ in $T$ because $T$ is connected. So, $\{u_3, v\} \subseteq N_T(u_2)$. For each $i \in \{2, 3\}$, let $T_i$ be the component tree of $T - \{e\}$ in which $u_i$ appears. If $u_4 \in V(T_3)$, then $u_4$ is a leaf of $T$ and hence adding the edge $\{u_2, u_4\}$ to $T - \{e\}$ clearly yields a spanning tree $\tilde{T}$ of $G$ with $|L(\tilde{T})| = |L(T)|$. So, we assume $u_4 \in V(T_2)$. If $u_1 \in V(T_3)$, then $u_4$ is a leaf of $T$ and hence adding the edge $\{u_1, u_4\}$ to $T - \{e\}$ clearly yields a spanning tree $\tilde{T}$ of $G$ with $|L(\tilde{T})| = |L(T)|$. Otherwise, $u_3$ is a leaf of $T$ and hence adding the edge $\{u_1, u_3\}$ to $T - \{e\}$ clearly yields a spanning tree $\tilde{T}$ of $G$ with $|L(\tilde{T})| = |L(T)|$. □

Lemma 8 Operation 9 is strongly safe.

Proof If $e \notin E(T)$, we are done. So, assume that $e \in E(T)$. Obviously, $\{u_4, u_5\} \cap L(T) \neq \emptyset$. Moreover, if for some $i \in \{4, 5\}$, $u_i \in L(T)$ and $\{u_2, u_i\} \in E(T)$, then the proof of Lemma 7 shows that $T$ can be transformed into a spanning tree $\tilde{T}$ such that $|L(\tilde{T})| \leq |L(T)|$ and $e \notin E(\tilde{T})$. Thus, we may assume that $u_5 \notin L(T)$, $\{u_1, u_4\} \in E(T)$, and $\{u_1, u_5\} \in E(T)$. Obviously, either $N_T(u_5) = \{u_2\}$ or $N_T(u_3) = \{u_1, u_2\}$. In the latter case, adding the edge $\{u_2, u_5\}$ to $T - \{e\}$ clearly yields a spanning tree $\tilde{T}$ of $G$, and $|L(\tilde{T})| = |L(T)|$ holds for $u_5 \in L(T)$. So, we assume the former case. Let $e' = \{u_1, u_5\}$. Then, adding the edges $\{u_1, u_3\}$ and $\{u_2, u_5\}$ to $T - \{e, e'\}$ clearly yields a spanning tree $\tilde{T}$ of $G$ with $|L(\tilde{T})| = |L(T)|$. □

Lemma 9 Operation 10 is strongly safe.

Proof Operation 10 is clearly strongly safe if $V(G) = V(K) \cup \{u, v\}$. So, we assume that $V(G) \neq V(K) \cup \{u, v\}$. Since $K$ is a connected component, the degree of each vertex $x \notin V(K)$ in $T - V(K)$ is $d_T(x)$ unless $x \in \{u, v\}$. Let $u \sim_T v$ be the path between $u$ and $v$ in $T$.

Let $S$ be the set of internal vertices of $u \sim_T v$. Since $K$ is a connected component of $G - \{u, v\}$, either $S \cap V(K) = \emptyset$ or $S \subseteq V(K)$. Obviously, we are done if $S = V(K)$. So, we assume that $S \cap V(K)$ is either empty or contains at least one but not all vertices of $K$. Then, $T - \{u, v\}$ has one or more component trees in which at least one vertex of $K$ appears. Let $T_1, \ldots, T_\ell$ be such component trees. For each $i \in \{1, \ldots, \ell\}$, $V(T_i) \subseteq V(K)$ because $K$ is a connected component of $G - \{u, v\}$. Moreover, if $V(T_i) \neq S$, then $L(T) \cap V(T_i) \neq \emptyset$. Since $V(T_i) \neq S$ for at least one $i \in \{1, \ldots, \ell\}$, $|L(T) \cap V(K)| \geq 1$. Furthermore, if $S \cap V(K) = \emptyset$, then $|L(T) \cap V(K)| \geq \ell$.

Case 1 $S$ is a nonempty proper subset of $V(K)$. Then, modifying $T - V(K)$ by adding the edges of $P$ yields a new spanning tree $\tilde{T}$ of $G$. Clearly, $L(\tilde{T}) \setminus L(T) \subseteq \{u, v\}$. Moreover, since $V(G) \neq V(K) \cup \{u, v\}$, it is impossible that $\{u, v\} \subseteq L(\tilde{T})$. So, $|L(\tilde{T}) \setminus L(T)| \leq 1$. Consequently, $|L(\tilde{T})| \leq |L(T)|$ because $|L(T) \cap V(K)| \geq 1$.

Case 2 $S \cap V(K) = \emptyset$. Then, both $u$ and $v$ are of degree at least 1 in $T - V(K)$. We assume that the degree of $u$ in $T - V(K)$ is at least as large as that of $v$ in $T - V(K)$; the other case is similar. Let $y$ be the neighbor of $u$ in $u \sim_T v$. It is possible that
y = v. Obviously, modifying $T - V(K)$ by adding the edges of $P$ and deleting the edge $\{u, y\}$ yields a new spanning tree $\tilde{T}$ of $G$. Clearly, $L(\tilde{T}) \setminus L(T) \subseteq \{u, y\}$. Thus, if $L(\tilde{T}) \setminus L(T) \neq \{u, y\}$, then $|L(\tilde{T})| \leq |L(T)|$ because $|L(T) \cap V(K)| \geq 1$. Moreover, if $\ell \geq 2$, then $|L(T) \cap V(K)| \geq 2$ and in turn $|L(\tilde{T})| \leq |L(T)|$. So, we may assume that $L(T) \setminus L(T) = \{u, y\}$ and $\ell = 1$. Then, the degree of $u$ in $T - V(K)$ is 1 and in turn so is $v$. Now, since $\ell = 1$ and $u \in L(\tilde{T}) \setminus L(T)$, $v$ is adjacent to no vertex of $K$ in $T$ and hence $v$ is a leaf of $T$. Therefore, no matter whether $y = v$ or not, $|L(T)| \leq |L(T)|$ because $|L(T) \cap V(K)| \geq 1$. 

\[
\Box
\]

**Lemma 10** Operation 11 is weakly safe.

**Proof** For each $i \in \{1, 2\}$, let $u'_i$ be the vertex in $N_G(u_i) \setminus \{u_{3-i}\}$. Possibly, $u'_1 = u'_2$. If $u'_1 \neq u'_2$, then $N_G(u_1u_2) = \{u'_1, u'_2\}$; otherwise, $N_G(u_1u_2) = \{u'_1\}$.

First, we want to show that $\text{opt}(G) \leq \text{opt}(G_1) + 1$. If $e \notin E(T)$, then $T$ contains both $\{u'_1, u_1\}$ and $\{u'_2, u_2\}$ and we can modify $T$ (without decreasing $|L(T)|$) by replacing the edge $\{u'_2, u_2\}$ with $e$. So, we can assume that $e \in E(T)$. Then, it is clear that modifying $T$ by merging $u_1$ and $u_2$ into a single vertex $u_1u_2$ yields a spanning tree of $G_1$ whose weight is $w(T) - 1$. Thus, $\text{opt}(G) \leq \text{opt}(G_1) + 1$.

Next, suppose that $T_1$ is a spanning tree of $G_1$. If $u'_1 = u'_2$, then $u_1u_2$ is a leaf of $T_1$ and its neighbor in $T_1$ is $u'_1$, and hence modifying $T_1$ by deleting the vertex $u_1u_2$ and adding the two edges $\{u'_1, u_1\}, \{u'_2, u_2\}$ yields a spanning tree of $G$ whose weight is $w(T_1) + 1$. So, we assume that $u'_1 \neq u'_2$. Clearly, at least one of $\{u'_1, u_1u_2\}$ and $\{u'_2, u_1u_2\}$ is an edge of $T_1$. If for exactly one $i \in \{1, 2\}, \{u'_i, u_1u_2\} \in E(T_1)$, then modifying $T_1$ by deleting the vertex $u_1u_2$ and adding the two edges $e, \{u'_i, u_i\}$ yields a spanning tree of $G$ whose weight is $w(T_1) + 1$. Otherwise, modifying $T_1$ by deleting the vertex $u_1u_2$ and adding the three edges $\{u'_1, u_1\}, e, \{u'_2, u_2\}$ yields a spanning tree of $G$ whose weight is $w(T_1) + 1$. 

\[
\Box
\]

Accordingly, we need to modify Step 1 of the algorithm by choosing $i$ from $\{1, 2, 8, 9, 10\}$ and also modify Step 2 by choosing $i$ from $\{3, 4, 11\}$. Obviously, after the modification, Steps 1 and 2 can be done in $O(n^2m)$ time.

## 5 Computing a preferred TFPCC $C$

In this section, we consider how to refine Step 4 because of Steps 1 and 3, we hereafter assume that $|V(G)| \geq 9$ and there is no $i \in \{1, \ldots, 4, 8, \ldots, 11\}$ such that Operation $i$ can be performed on $G$. Then, we can prove the next lemma:

**Lemma 11** Suppose that $C$ is a cycle of $G$ with $|C| \leq 8$. Let $A$ be the set of ports of $C$. Then, the following statements hold.

1. $|A| \geq 2$.
2. If $|A| = 2$, then the two vertices in $A$ are not adjacent in $C$ and $|C| \neq 5$.
3. If $|A| = 2$ and $|C| = 4$, then $G[V(C)]$ and $C$ are the same graph.

**Proof** We prove the statements separately as follows.

**Statement 1** Since $G$ is connected and $|C| < 9 \leq |V(G)|$, $|A| \geq 1$. Moreover, since Operation 4 cannot be performed on $G$, $|A| \geq 2$. 

\[\copyright\ Springer\]
Lemma 12 Suppose that \(|A| = 2\). Then, the two vertices in \(A\) cannot be adjacent in \(C\), because otherwise Operation 10 could be performed on \(G\). For a contradiction, assume that \(|C| = 5\). Suppose that \(u_1, \ldots, u_5\) are the vertices of a 5-cycle of \(C\) and appear in \(C\) clockwise in this order. Since the two vertices in \(A\) are not adjacent in \(C\), we may assume that \(A = \{u_1, u_3\}\). If \(\{u_2, u_4\} \in E(G)\) or \(\{u_2, u_5\} \in E(G)\), then Operation 10 can be performed on \(G\), a contradiction. So, we assume that \(\{u_2, u_4\} \notin E(G)\) and \(\{u_2, u_5\} \notin E(G)\). If \(\{u_1, u_4\} \in E(G)\) or \(\{u_3, u_5\} \in E(G)\), then Operation 10 can be performed on \(G\), a contradiction. Thus, we may further assume that \(\{u_1, u_4\} \notin E(G)\) and \(\{u_3, u_5\} \notin E(G)\). Now, \(\{u_4, u_5\} \in E(G)\), \(d_G(u_4) = 2\), and \(d_G(u_5) = 2\). Hence, Operation 11 can be performed on \(G\), a contradiction.

Statement 2 Suppose that \(|A| = 2\) and \(|C| = 4\). The two vertices in \(A\) are not adjacent in \(C\) by Statement 2, and hence \(G[V(C)]\) and \(C\) are the same graph because otherwise Operation 10 could be performed on \(G\).

To refine Step 4, our idea is to compute \(C\) as a preferred TFPCC of \(G\). Before defining what the word “preferred” means here, we need to prove a lemma. For ease of explanation, we assume, without loss of generality, that there is a linear order (denoted by \(\prec\)) on the vertices of \(G\).

Lemma 12 Suppose that \(u_1\) and \(u_3\) are two vertices of \(G\) such that \(u_1 \prec u_3\) and Condition C1 below holds. Then, \(G\) has an OST in which \(u_1\) or \(u_3\) is a leaf. Consequently, \(G\) has an OST in which \(u_1\) is a leaf.

C1. \(V(G)\) \(\setminus \{u_1, u_3\}\) contains two vertices \(u_2\) and \(u_4\) such that \(N_G(u_1) = N_G(u_3) = \{u_2, u_4\}\).

Proof If \(u_1\) is a leaf of \(T\), then we are done. So, assume that \(u_1\) is not a leaf of \(T\). Since Condition C1 holds, \(u_3\) is clearly a leaf of \(T\) and we can modify \(T\) (without decreasing \(w(T)\)) by switching \(u_1\) and \(u_3\) so that \(u_1\) becomes a leaf in \(T\).

If Condition C1 in Lemma 12 holds for \(u_1\) and \(u_3\), we refer to \(u_2\) and \(u_4\) as the boundary points of the pair \(p = (u_1, u_3)\), and refer to the edges incident to \(u_1\) or \(u_3\) as the supports of \(p\).

Let \(\Pi\) be the set of pairs \((u_1, u_3)\) of vertices in \(G\) satisfying Condition C1. It is worth pointing out that for each \(p \in \Pi\) and each boundary point \(u\) of \(p\), \(d_G(u) \geq 3\) because otherwise Operation 4 could be performed on \(G\).

Lemma 13 No two pairs in \(\Pi\) share a support.

Proof Obviously, for two pairs in \(\Pi\) to share a support, they have to share their boundary points. However, no two pairs in \(\Pi\) can share their boundary points, because otherwise Operation 9 could be performed on \(G\). So, no two pairs in \(\Pi\) share a support.

Lemma 14 \(G\) has an OST in which \(u_1\) is a leaf for each \((u_1, u_3) \in \Pi\).

Proof By Lemma 12, we can assume that for every \(p = (u_1, u_3) \in \Pi\), \(d_T(u_1) \leq 1\). In a nutshell, the proof of Lemma 12 shows that even if \(T\) is an OST with \(d_T(u_1) \geq 2\), we can modify \(T\) without decreasing \(w(T)\) so that \(d_T(u_1) \leq 1\). Indeed, the modification only uses the supports of \(p\). Now, by Lemma 13, a similar modification can be done independently for each other \(p' \in \Pi\). Therefore, the lemma holds.
Now, we are ready to make two definitions. Let $C$ be a TFPCC of $G$. $C$ is special if for every pair $(u_1, u_3) \in \Pi$, $d_C(u_1) \leq 1$. $C$ is preferred if $C$ is special and $|E(C)|$ is maximized over all special TFPCCs of $G$.

**Lemma 15** If $C$ is a preferred TFPCC of $G$, then $\text{opt}(G) \leq |E(C)|$.

**Proof** By Lemma 14, $G$ has an OST $\tilde{T}$ such that for each $(u_1, u_3) \in \Pi$, $d_{\tilde{T}}(u_1) = 1$. So, by Lemma 5, we can construct a path cover $\mathcal{P}$ of $G$ with $|E(\mathcal{P})| \geq w(\tilde{T})$ such that $d_P(u_1) \leq 1$ for every $(u_1, u_3) \in \Pi$. Thus, $\mathcal{P}$ is a special TFPCC of $G$. Consequently, if $C$ is a preferred TFPCC of $G$, then $\text{opt}(G) = w(\tilde{T}) \leq |E(\mathcal{P})| \leq |E(C)|$. $\square$

**Lemma 16** We can compute a preferred TFPCC $C$ of $G$ in $t(2n, 2m)$ time.

**Proof** We construct a new graph $G'$ from $G$ by adding a new vertex $x_p$ and the edge $(u_1, x_p)$ for each pair $p = (u_1, u_3) \in \Pi$. Obviously, if $C^*$ is a preferred TFPCC of $G$, then adding the edges $(x_p, u_1)$ with $p = (u_1, u_3) \in \Pi$ to $C^*$ yields a TFPCC $C'$ of $G'$ with $|E(C')| = |E(C^*)| + |\Pi|$. Clearly, $C'$ is still a maximum TFPCC of $G'$ after the modification. So, we can repeatedly modify $C'$ in this way until $d_{C'}(x_p) = 1$ for every $p = (u_1, u_3) \in \Pi$. $C'$ is now a maximum TFPCC of $G'$ such that for every $p = (u_1, u_3) \in \Pi$, $d_{C'}(x_p) = 1$. Finally, we obtain $C$ from $C'$ by deleting the edge $(x_p, u_1)$ for each $p = (u_1, u_3) \in \Pi$. Clearly, $|E(C)| = |E(C')| - |\Pi| \geq |E(C^*)|$. Therefore, $C$ is a preferred TFPCC of $G$. Since $|V(G')| \leq 2|V(G)|$ and $|E(G')| \leq 2|E(G)|$, the lemma holds. $\square$

Recall that $t(n, m) = O(n^2m^2)$ (Hartvigsen 1984). So, Lemma 16 ensures that after modifying Step 4 by computing $C$ as a preferred TFPCC of $G$, Step 4 can still be done in $t(n, m)$ time.

**6 Preprocessing $C$**

In this section, we consider how to refine Step 5. So, suppose that we have computed a preferred TFPCC $C$ of $G$ as in Lemma 16. To refine Step 5, we repeatedly perform not only Operations 5 through 7 but also the following three operations on $C$ until none of the six is applicable.

**Operation 12** If a cycle $C_1$ of $C$ has an edge $e_1 = \{u_1, u'_1\}$ and another cycle or path component $C_2$ of $C$ has an edge $e_2 = \{u_2, u'_2\}$ such that $e = \{u_1, u_2\} \in E(G)$ and $e' = \{u'_1, u'_2\} \in E(G)$, then combine $C_1$ and $C_2$ into a single cycle or path by replacing $e_1$ and $e_2$ with $e$ and $e'$.

**Operation 13** If an endpoint $u_1$ of a path component $P_1$ of $C$ is adjacent to an endpoint $u_2$ of another path component $P_2$ of $C$ in $G$, then combine $P_1$ and $P_2$ into a single path by adding the edge $\{u_1, u_2\}$.
Operation 14 If \( e = \{u, v\} \) is an edge of a path component of \( C \) such that for some isolated vertex \( x \) of \( C \), \( \{u, x\} \in E(G) \) and \( \{v, x\} \in E(G) \), then replace \( e \) by the edges \( \{u, x\} \) and \( \{v, x\} \).

Lemma 17 Immediately after the refined preprocessing step, the following statements hold:

1. \( C \) is a TFPCC of \( G \) and has at least \( \text{opt}(G) \) edges.
2. If a path component \( P \) of \( C \) is of length at most 3, then \( P \) is alive.
3. If an endpoint \( v \) of a path component \( P \) of \( C \) is a port of \( P \), then each vertex in \( N_G(v) \setminus V(P) \) is an internal vertex of a path component \( Q \) of \( C \) with \( |Q| \geq 2|P|+2 \).
4. No pair \( (u_1, u_3) \in \Pi \) satisfies that \( u_1 \) appears in a cycle of \( C \).
5. If a dead path component \( P \) of \( C \) is of length 4, then both endpoints of \( P \) are leaves in \( G \).
6. Each 4-cycle \( C \) of \( C \) has at least three ports.

Proof A short cycle is a cycle of length at most 7. We prove the statements separately as follows.

Statement 1 Before the refined preprocessing, \( C \) has at least \( \text{opt}(G) \) edges by Lemma 15 and is a TFPCC of \( G \). Since Operation \( i \) does not decrease the number of edges in \( C \) or creates a new short cycle or a vertex of degree larger than 2 in \( C \) for each \( i \in \{5, 6, 7, 12, 13, 14\} \), Statement 1 holds.

Statement 2 Same as that of Statement 2 in Lemma 6.

Statement 3 Suppose that an endpoint \( v \) of a path component \( P \) of \( C \) is a port. Consider an arbitrary \( u \in N_G(v) \setminus V(P) \). Since neither Operation 6 nor Operation 13 can be performed on \( C \), \( u \) is an internal vertex of a path component \( Q \) of \( C \). Let \( u_1 \) and \( u_2 \) be the endpoints of \( Q \). For each \( i \in \{1, 2\} \), let \( Q_i \) be the path from \( u \) to \( u_i \) in \( P \). Then, \( |Q| = |Q_1| + |Q_2| \). Moreover, since Operation 7 cannot be applied on \( C \), \( |P| + |Q_i| + 1 \leq |Q| \) for each \( i \in \{1, 2\} \). Thus, \( 2|P| + 2 \leq |Q| \).

Statement 4 Before the refined preprocessing, no pair \( (u_1, u_3) \in \Pi \) satisfies that \( u_1 \) appears in a cycle of \( C \) because \( C \) is a preferred TFPCC of \( G \). Moreover, if Operation \( i \) creates a new cycle \( C \) in \( C \) for some \( i \in \{5, 6, 7, 12, 13, 14\} \), then \( i = 12 \) and \( C \) is obtained by merging two shorter cycles in \( C \). Thus, Statement 4 holds.

Statement 5 Let \( P \) be a dead path component of \( C \) with \( |P| = 4 \). Suppose that \( u_1, \ldots, u_5 \) are the vertices of \( P \) and they appear in \( P \) in this order. If all internal vertices of \( P \) are ports, then both \( u_1 \) and \( u_5 \) are leaves of \( G \) (and we are done), because otherwise Operation 5 could be performed on \( C \). Moreover, if at most one internal vertex of \( P \) is a port, then \( G \) would be disconnected or Operation 4 could be performed on \( G \), a contradiction. So, we assume that exactly two internal vertices of \( P \) are ports. Now, if \( \{i, j\} = \{2, 4\} \), then \( \{u_1, u_5\} \notin E(G), \{u_1, u_3\} \notin E(G) \), and \( \{u_5, u_3\} \notin E(G) \) (because otherwise Operation 5 could be performed on \( C \), and in turn both \( u_1 \) and \( u_5 \) are leaves of \( G \) (and we are done) because otherwise Operation 8 or 9 could be performed on \( G \). Thus, we may assume that \( i = 2 \) and \( j = 3 \). Then, since Operation 5 cannot be performed on \( C \), \( u_1 \) is a leaf of \( G \) and \( \{u_1, u_2\} \cap N_G(u_5) = \emptyset \). For the same reason, \( \{u_3, u_5\} \notin E(G) \) or \( \{u_2, u_4\} \notin E(G) \). Indeed, \( \{u_2, u_4\} \in E(G) \) because otherwise the edge \( \{u_2, u_3\} \) would be deleted by Operation 2 or 3. Therefore, \( \{u_3, u_5\} \notin E(G) \) and in turn \( u_5 \) is also a leaf of \( G \).
Statement 6 Let \( C \) be a 4-cycle in \( C \), and \( A \) be the set of ports of \( C \). Further let \( u_1, \ldots, u_4 \) be the vertices of \( C \) and assume that they appear in \( C \) clockwise in this order. By Lemma 11, \( |A| \geq 2 \). For a contradiction, assume that \( |A| = 2 \). Then, by Statement 3 in Lemma 11, \( A = \{u_1, u_3\} \) or \( A = \{u_2, u_4\} \). We may assume that \( A = \{u_2, u_4\} \) and \( u_1 < u_3 \). Then, \( N_G(u_1) = N_G(u_3) = \{u_2, u_4\} \) by Statement 3 in Lemma 11, and in turn \( (u_1, u_3) \in \Pi \). Since the refined preprocessing of \( C \) does not introduce a new short cycle, \( C \) is a cycle in \( C \) even before the refined preprocessing. However, this contradicts the fact that \( C \) is a preferred TFPCC of \( G \) before the refined preprocessing.

\[ \square \]

Obviously, the refined preprocessing (i.e., Step 5) can be done in \( O(nm) \) time.

7 Transforming \( C \) into a spanning tree

In this section, we consider how to refine Step 6. So, suppose that we have just performed the refined preprocessing on \( C \) as in Sect. 6. Let \( \Gamma \) be the set of (ordered) pairs \( (P, Q) \) of path components of \( \hat{C} \) such that \( |P| \geq 1 \) and some endpoint \( v \) of \( P \) is adjacent to a vertex \( u \) of \( Q \) in \( G \). Note that \( d_C(u) = 2 \) and \( 2|P| + 2 \leq |Q| \) by Statement 3 in Lemma 17. Suppose that we obtain a subset \( \Gamma' \) of \( \Gamma \) as follows.

- For each path component \( P \) of \( \hat{C} \) such that there are two or more path components \( Q \) of \( \hat{C} \) with \( (P, Q) \in \Gamma \), delete all but one pair \( (P, Q) \) from \( \Gamma \).

Now, consider an auxiliary digraph \( D \) such that the vertices of \( D \) one-to-one correspond to the path components \( P \) of \( \hat{C} \) with \( |P| \geq 1 \) and the arcs of \( D \) one-to-one correspond to the pairs in \( \Gamma' \). By Statement 3 in Lemma 17, \( D \) is a rooted forest (in which each leaf is of in-degree 0, each root is of out-degree 0, and each vertex is of out-degree at most 1).

To transform \( C \) into a spanning tree of \( G \), the idea is to modify \( C \) in three stages. \( C \) is initially a TFPCC of \( G \) and we will always keep \( C \) being a TFTCC of \( G \). For each \( i \in \{1, 2, 3\} \), we use \( C_i \) to denote the \( C \) immediately after the \( i \)-th stage. For convenience, we use \( C_0 \) to denote the \( C \) immediately before the first stage. Moreover, for each \( i \in \{1, 2, 3\} \) and each connected component \( C \) of \( C_i \), we use \( b(C) \) to denote the number of edges \( \{u, v\} \in E(C_0) \) such that \( \{u, v\} \subseteq V(C) \).

In the first stage, we modify \( C \) by performing the following step:

1. For each pair \( (P, Q) \in \Gamma' \), add an arbitrary \( \{u, v\} \in E(G) \) to \( \hat{C} \) such that \( u \) is an endpoint of \( P \) and \( v \) appears in \( Q \).

Lemma 18 Each connected component of \( C_1 \) that is not a path or cycle is a tree \( \hat{T} \) satisfying Condition C2 below:

\begin{align*}
\text{C2. } & b(\hat{T}) \geq 5, |L(\hat{T})| \leq b(\hat{T}) - 2, \text{ and } w(\hat{T}) \geq \frac{4}{5}b(\hat{T}).
\end{align*}

Proof Let \( \hat{T} \) be a connected component of \( C_1 \) that is not a path or cycle. Obviously, \( \hat{T} \) can be obtained from a tree component \( \hat{T}_D \) of \( D \) by replacing each vertex of \( \hat{T}_D \) with the corresponding path component of \( C \) and replacing each arc of \( \hat{T}_D \) corresponding to a pair \( (P, Q) \in \Gamma' \) with an edge \( \{v, u\} \in E(G) \) such that \( v \) is an endpoint of \( P \) and \( u \) appears in \( Q \). Thus, \( \hat{T} \) is clearly a tree.
We next prove that \( \hat{T} \) satisfies Condition C2 by induction on the number of arcs in \( \hat{T}_D \). Clearly, \( \hat{T}_D \) has at least one edge. In the base case, \( \hat{T}_D \) has only one arc. Let \( (P, Q) \) be the pair in \( \Gamma' \) corresponding to the arc. \( \hat{T} \) is obtained from \( P \) and \( Q \) by connecting them with an edge \( \{v, u\} \in E(G) \) such that \( v \) is an endpoint of \( P \) and \( u \) appears in \( Q \). Thus, \( w(\hat{T}) = |P| + |Q| - 1, |L(\hat{T})| = 3, \) and \( b(\hat{T}) = |P| + |Q| \). Hence, by Statement 3 in Lemma 17, \( b(\hat{T}) \geq 3|P| + 2 \geq 5 \). Therefore, \( |L(\hat{T})| \leq b(\hat{T}) - 2 \) and \( w(\hat{T}) = 1 - \frac{1}{b(\hat{T})} \geq \frac{4}{5} \). This shows that \( \hat{T} \) satisfies Condition C2 in the base case.

Now, assume that \( \hat{T}_D \) has at least two arcs. Consider an arbitrary \( (P, Q) \in \Gamma' \) such that the vertex \( \alpha \) of \( D \) corresponding to \( P \) is a leaf of \( \hat{T}_D \). Let \( \hat{T}'_D \) be obtained from \( \hat{T}_D \) by deleting \( \alpha \), and \( \hat{T}' \) be obtained from \( \hat{T} \) by deleting the vertices of \( P \). Since \( \hat{T}'_D \) has one fewer arc than \( \hat{T}_D \), the inductive hypothesis implies that \( b(\hat{T}') \geq 5 \), \( |L(\hat{T}')| \leq b(\hat{T}') - 2 \), and \( w(\hat{T}') \geq \frac{4}{5} b(\hat{T}') \). Obviously, \( b(\hat{T}) = b(\hat{T}') + |P|, |L(\hat{T})| = |L(\hat{T}')| + 1, \) and \( w(\hat{T}) = w(\hat{T}') + |P| \). Since \( |P| \geq 1 \), it is now easy to verify that \( \hat{T} \) satisfies Condition C2.

Hereafter, a connected component of \( C \) is good if it is a tree \( \hat{T} \) satisfying Condition C2 in Lemma 18 or Condition C3 below, while it is bad otherwise.

**Lemma 19** Suppose that \( C \) is a bad connected component of \( C_1 \). Then, \( C \) is a cycle of length at least 4, a 0-path, or a 4-path whose endpoints are leaves of \( G \). Moreover, if \( C \) is a 0-path, then the unique vertex \( v \in V(C) \) satisfies that each \( v \in N_G(u) \) is an internal vertex of a tree component of \( C_1 \) and no two vertices in \( N_G(u) \) are adjacent in \( C_1 \).

**Proof** Since \( C \) is bad, Lemma 18 ensures that \( C \) is a path or cycle and in turn is a connected component of \( C_0 \). Indeed, \( C \) cannot be a path of length at least 5, because otherwise \( C \) would satisfy Condition C2. Now, by Lemma 17, \( C \) is a cycle of length at least 4, a 0-path, or a 4-path whose endpoints are leaves of \( G \).

Suppose that \( C \) is a 0-path. Then, \( C \) is also 0-path in \( C_0 \). Let \( u \) be the unique vertex in \( C \). Consider an arbitrary \( v \in N_G(u) \). Since Operation 13 cannot be performed on \( C_0 \), \( v \) is not a leaf of a tree component of \( C_1 \). Moreover, since Operation 6 cannot be performed on \( C_0 \), \( v \) does not appear in a cycle of \( C_1 \). Furthermore, since Operation 14 cannot be performed on \( C_0 \), no two vertices in \( N_G(u) \) are adjacent in \( C_1 \).

We next want to define several operations on \( C \) none of which will produce a new cycle or a new bad connected component in \( C \). An operation on \( C \) is good if it either just connects two or more connected components of \( C \) into a single good connected component, or modifies a good connected component of \( C \) so that it has more internal vertices (and hence remains good).

In the second stage, we modify \( C \) by repeatedly performing the following operations on \( C \) until none of them is applicable.

**Operation 15** If \( C \) has two cycles \( C_1 \) and \( C_2 \) such that \( |C_1| + |C_2| \geq 10 \) and some edge \( e = \{v_1, v_2\} \) of \( G \) satisfies \( v_1 \in V(C_1) \) and \( v_2 \in V(C_2) \), then connect \( C_1 \) and \( C_2 \) into a single path \( T \) by deleting one edge incident to \( v_1 \) in \( C_1 \), deleting one edge incident to \( v_2 \) in \( C_2 \), and adding the edge \( e \).
**Operation 16** If $C$ has a cycle $C_1$ of length at least 5 and a good connected component $C_2$ such that some edge $e = \{v, u\}$ of $G$ satisfies $v \in V(C_1)$ and $u \in V(C_2)$, then connect $C_1$ and $C_2$ into a single tree $T$ by deleting one edge incident to $v$ in $C_1$ and adding the edge $e$.

**Operation 17** If $C$ has a cycle $C$ of length at least 6 and a 4-path component $P$ such that some edge $e = \{v, u\}$ of $G$ satisfies $v \in V(C)$ and $u \in V(P)$, then connect $C$ and $P$ into a single tree $T$ by deleting one edge incident to $v$ in $C$ and adding the edge $e$.

**Operation 18** If $C$ has a 0-path component $P$ whose unique vertex $u$ has two neighbors $v_1$ and $v_2$ in $G$ such that $v_1$ and $v_2$ fall into different connected components $C_1$ and $C_2$ of $C$, then connect $P$, $C_1$, and $C_2$ into a single connected component $T$ by adding the edges $\{u, v_1\}$ and $\{u, v_2\}$.

**Operation 19** If $C$ has a good connected component $C_1$ and another connected component $C_2$ such that some leaf $u$ of $C_1$ is adjacent to a vertex $v$ of $C_2$ in $G$, then connect $C_1$ and $C_2$ into a single tree component $T$ by deleting one edge incident to $v$ in $C_2$ if $C_2$ is a cycle, and further adding the edge $\{u, v\}$.

**Operation 20** If a cycle $C$ of $\bar{C}$ has an edge $e = \{v_1, v_2\}$ such that some $u_1 \in N_G(v_1) \setminus V(C)$ and some $u_2 \in N_G(v_2) \setminus V(C)$ fall into different connected components $C_1$ and $C_2$ of $\bar{C}$ other than $C$, then connect $C$, $C_1$, and $C_2$ into a single tree component $T$ by deleting $e$, deleting one edge incident to $u_1$ if $C_1$ is a cycle, deleting one edge incident to $u_2$ if $C_2$ is a cycle, and adding the edges $\{v_1, u_1\}$ and $\{v_2, u_2\}$.

**Operation 21** If a good connected components $C$ of $\bar{C}$ is not a Hamiltonian path of $G$ but is a dead path whose endpoints are adjacent in $G$, then choose an arbitrary port $u$ of $C$, modify $C$ by adding the edge of $G$ between the endpoints of $C$ and deleting one edge incident to $u$ in $C$, and further perform **Operation 19**.

**Operation 22** If a good connected component $C$ of $\bar{C}$ is not a path but has two leaves $u$ and $v$ with $\{u, v\} \in E(G)$, then modify $C$ by first finding an arbitrary vertex $x$ on the path $P$ between $u$ and $v$ in $C$ with $d_C(x) \geq 3$, then deleting one edge incident to $x$ in $P$, and further adding the edge $\{u, v\}$.

**Operation 23** If $\bar{C}$ has a 0-path component $C_1$, a 4-path component $P$, and a connected component $C_2$ other than $C_1$ and $P$ such that the center vertex $u_3$ of $P$ is adjacent to a vertex $x$ of $C_2$ in $G$ and the unique vertex $v$ of $C_1$ is adjacent to the other two internal vertices $u_2$ and $u_4$ of $P$ (than $u_3$) in $G$, then connect $C_1$, $P$, and $C_2$ into a single connected component $T$ by deleting the edge $\{u_2, u_3\}$, deleting one edge incident to $x$ if $C_2$ is a cycle, and adding the edges $\{v, u_2\}$, $\{v, u_4\}$, $\{u_3, x\}$.

In the following proofs of Lemmas 20 through 28, $\hat{T}$ denotes the new connected component of $C$ created by the corresponding operation.

**Lemma 20** Operation 15 is good.

*Proof* Obviously, $w(\hat{T}) = |C_1| + |C_2| - 2$, $b(\hat{T}) = |C_1| + |C_2| \geq 10$, and $|L(\hat{T})| = 2$. Thus, $\hat{T}$ is good.

$\square$
Lemma 21  Operation 16 is good.

Proof  Obviously, \(w(\hat{T}) \geq w(C_2) + |C_1| - 1 \geq \frac{4}{5} b(C_2) + |C_1| - 1, b(\hat{T}) = b(C_2) + |C_1| \geq b(C_2) + 5, \) and \(|L(\hat{T})| \leq |L(C_2)| + 1\). Thus, \(\hat{T}\) is good.

Lemma 22  Operation 17 is good.

Proof  Obviously, \(w(\hat{T}) = |C| + 2, b(\hat{T}) = |C| + 4 \geq 10, \) and \(|L(\hat{T})| = 3\). Thus, \(\hat{T}\) is good.

Lemma 23  Operation 18 is good.

Proof  Since Operation 6 cannot be applied on \(C_0\), neither \(C_1\) nor \(C_2\) is a cycle. Hence, both \(C_1\) and \(C_2\) are trees and in turn \(\hat{T}\) is a tree. To show that \(\hat{T}\) is good, we distinguish three cases as follows.

Case 1  Both \(C_1\) and \(C_2\) are good. In this case, \(w(\hat{T}) \geq w(C_1) + w(C_2) + 1 \geq \frac{4}{5} b(C_1) + \frac{4}{5} b(C_2) + 1, b(\hat{T}) = b(C_1) + b(C_2) \geq 8, \) and \(|L(\hat{T})| \leq |L(C_1)| + |L(C_2)|\). Thus, \(\hat{T}\) is clearly good.

Case 2  One of \(C_1\) and \(C_2\) is good. W.l.o.g., we assume that \(C_1\) is good and \(C_2\) is bad. Then, by Lemma 19, \(C_2\) is either a 0-path or a 4-path whose endpoints are leaves of \(G\). The former case is impossible, because Operation 13 cannot be performed on \(C_0\). In the latter case, \(w(\hat{T}) \geq w(C_1) + 4 \geq \frac{4}{5} b(C_1) + 4, b(\hat{T}) = b(C_1) + 4 \geq 8, \) and \(|L(\hat{T})| \leq |L(C_1)| + 2\), implying that \(\hat{T}\) is good.

Case 3  Both \(C_1\) and \(C_2\) are bad. In this case, both \(C_1\) and \(C_2\) are 4-paths whose endpoints are leaves of \(G\), because Operation 13 cannot be performed on \(C_0\). So, \(w(\hat{T}) = 7, b(\hat{T}) = 8, \) and \(|L(\hat{T})| = 4\), implying that \(\hat{T}\) is good.

Lemma 24  Operation 19 is good.

Proof  \(\hat{T}\) is clearly a tree. To show that \(\hat{T}\) is good, we distinguish three cases as follows.

Case 1  \(C_2\) is a cycle. In this case, \(w(\hat{T}) = w(C_1) + |C_2| \geq \frac{4}{5} b(C_1) + |C_2|, b(\hat{T}) = b(C_1) + |C_2| \geq 8, \) and \(|L(\hat{T})| = |L(C_1)|\). So, \(\hat{T}\) is clearly good.

Case 2  \(C_2\) is good. In this case, \(w(\hat{T}) \geq w(C_1) + w(C_2) + 1 \geq \frac{4}{5} b(C_1) + \frac{4}{5} b(C_2) + 1, b(\hat{T}) = b(C_1) + b(C_2) \geq 8, \) and \(|L(\hat{T})| = |L(C_1)| + |L(C_2)| - 1\). So, \(\hat{T}\) is clearly good.

Case 3  \(C_2\) is bad but not a cycle. In this case, Lemma 19 ensures that \(C_2\) is either a 0-path or a 4-path whose endpoints are leaves of \(G\). In the latter case, \(w(\hat{T}) \geq w(C_1) + 4 \geq \frac{4}{5} b(C_1) + 4, b(\hat{T}) = b(C_1) + 4 \geq 8, \) and \(|L(\hat{T})| = |L(C_1)| + 1\), implying that \(\hat{T}\) is clearly good. So, we assume the former case. If \(C_1\) satisfies Condition C2, then \(w(\hat{T}) = w(C_1) + 1 \geq \frac{4}{5} b(C_1) + 1, b(\hat{T}) = b(C_1) \geq 5, \) and \(|L(\hat{T})| = |L(C_1)|\), implying that \(\hat{T}\) is good. Otherwise, \(b(\hat{T}) = b(C_1) = 4, w(\hat{T}) = w(C_1) + 1 \geq b(C_1) + 1 > b(\hat{T}), \) and \(|L(\hat{T})| = |L(C_1)|\), implying that \(\hat{T}\) is good.

Lemma 25  Operation 20 is good.

Proof  \(\hat{T}\) is clearly a tree. To show that \(\hat{T}\) is good, we distinguish three cases as follows.

Case 1  Both \(C_1\) and \(C_2\) are cycles. In this case, \(w(\hat{T}) = |C| + |C_1| + |C_2| - 2, b(\hat{T}) = |C| + |C_1| + |C_2| \geq 12, \) and \(|L(\hat{T})| = 2\). Thus, \(\hat{T}\) is clearly good.
Case 2 One of \( C_1 \) and \( C_2 \) is a cycle. W.l.o.g., we assume that \( C_2 \) is a cycle. If \( C_1 \) is good, then \( w(\hat{T}) \geq w(C_1) + |C_1| + |C_2| - 1 \geq \frac{4}{5}b(C_1) + |C_1| + |C_2| - 1, \)
\( b(\hat{T}) = b(C_1) + |C_1| + |C_2| \geq b(C_1) + 8 \geq 12, \)
and \( |L(\hat{T})| \leq |L(C_1)| + 1, \) implying that \( \hat{T} \) is good. So, assume that \( C_1 \) is bad. Then, by Lemma 19, \( C_1 \) is a 0-path or 4-path whose endpoints are leaves of \( G \). Indeed, \( C_1 \) is not a 0-path, because Operation 6 cannot be performed on \( C_0 \). Thus, \( w(\hat{T}) = |C| + |C_2| + 2, b(\hat{T}) = |C| + |C_2| + 4 \geq 12, \)
and \( |L(\hat{T})| = 3 \). Hence, \( \hat{T} \) is good.

Case 3 Neither \( C_1 \) nor \( C_2 \) is a cycle. If both \( C_1 \) and \( C_2 \) are good, then \( w(\hat{T}) \geq |C| + w(C_1) + w(C_2) \geq |C| + \frac{4}{5}b(C_1) + \frac{4}{5}b(C_2), b(\hat{T}) = |C| + b(C_1) + b(C_2) \geq b(C_1) + b(C_2) + 4 \geq 12, \)
and \( |L(\hat{T})| \leq |L(C_1)| + |L(C_2)|, \) implying that \( \hat{T} \) is good. Similarly, if both \( C_1 \) and \( C_2 \) are bad, then both of them are 4-paths whose endpoints are leaves of \( G \) and in turn \( w(\hat{T}) = |C| + 6, b(\hat{T}) = |C| + 8 \geq 12, \)
and \( |L(\hat{T})| = 4, \) implying that \( \hat{T} \) is good. So, we may assume that \( C_1 \) is good but \( C_2 \) is bad. Then, \( C_2 \) is a 4-path whose endpoints are leaves of \( G \). Hence, \( w(\hat{T}) \geq |C| + w(C_1) + 3 \geq |C| + \frac{4}{5}b(C_1) + 3, b(\hat{T}) = |C| + b(C_1) + 4 \geq b(C_1) + 8 \geq 12, \)
and \( |L(\hat{T})| \leq |L(C_1)| + 2. \) Therefore, \( \hat{T} \) is good.

\[\square\]

**Lemma 26** Operation 21 is good.

**Proof** By Lemma 24, Operation 21 is clearly good.

\[\square\]

**Lemma 27** Operation 22 is good.

**Proof** The operation clearly decreases the number of leaves in \( C \) by 1, and hence good.

\[\square\]

**Lemma 28** Operation 23 is good.

**Proof** \( \hat{T} \) is clearly a tree. To show that \( \hat{T} \) is good, we distinguish three cases as follows.

Case 1 \( C_2 \) is a cycle. In this case, \( w(\hat{T}) = |C_2| + 3, b(\hat{T}) = |C_2| + 4 \geq 8, \)
and \( |L(\hat{T})| = 3 \). So, \( \hat{T} \) is clearly good.

Case 2 \( C_2 \) is good. In this case, \( w(\hat{T}) \geq w(C_2) + 4 \geq \frac{4}{5}b(C_2) + 4, b(\hat{T}) = b(C_2) + 4 \geq 8, \)
and \( |L(\hat{T})| \leq |L(C_2)| + 2. \) So, \( \hat{T} \) is clearly good.

Case 3 \( C_2 \) is bad. In this case, \( C_2 \) is either a 0-path or a 4-path whose endpoints are leaves of \( G \). In the former case, \( w(\hat{T}) = 4, b(\hat{T}) = 4, \) and \( |L(\hat{T})| = 3 \), implying that \( \hat{T} \) is clearly good. In the latter case, \( w(\hat{T}) = 7, b(\hat{T}) = 8, \) and \( |L(\hat{T})| = 4, \) implying that \( \hat{T} \) is clearly good.

\[\square\]

We next show that the above operations lead to a number of useful properties of \( C_2 \).

**Lemma 29** Each 4-cycle of \( C_2 \) is adjacent to at most one other connected component of \( C_2 \) in \( G \).

**Proof** Let \( C \) be a 4-cycle in \( C_2 \). Further let \( v_1, \ldots, v_4 \) be the vertices of \( C \) and assume that they appear in \( C \) clockwise in this order. By Statement 6 in Lemma 17, \( C \) has at least three ports. Without loss of generality, we may assume that \( v_1 \) through \( v_3 \) are ports of \( C \). Since Operation 20 cannot be performed on \( C_2 \), there is a unique connected component \( C' \) in \( C_2 \) such that \( N_G(\{v_1, v_2\}) \setminus V(C) \) is a nonempty subset of \( V(C') \). For
the same reason, \( N_G((v_2, v_3)) \setminus V(C) \) is a nonempty subset of \( V(C') \). Moreover, if \( v_4 \) is also a port of \( C \), then for the same reason, \( N_G((v_3, v_4)) \setminus V(C) \) is a nonempty subset of \( V(C') \). Therefore, in any case, \( N_G(C) \setminus V(C) \subseteq V(C') \) and hence \( C \) is adjacent to only \( C' \) in \( G \). □

**Lemma 30** No two 4-cycles of \( C_2 \) are adjacent in \( G \).

**Proof** For a contradiction, assume that two 4-cycles \( C_1 \) and \( C_2 \) of \( C_2 \) are adjacent in \( G \). Then, by Lemma 29, \( G[V(C_1) \cup V(C_2)] \) is a connected component of \( G \). However, this is impossible because \( G \) is connected and \( |V(G)| \geq 9 \). □

**Lemma 31** No 4-cycle \( C \) of \( C_2 \) is adjacent to a 4-path component of \( C_2 \) in \( G \).

**Proof** For a contradiction, assume that a 4-cycle \( C \) of \( C_2 \) is adjacent to a 4-path component \( P \) of \( C_2 \) in \( G \). Let \( v_1, \ldots, v_4 \) be the vertices of \( C \) and assume that they appear in \( C \) clockwise in this order. Let \( B \) be the set of all \( u \in V(G) \setminus V(C) \) such that for some \( v_i \in V(C), \{u, v_i\} \in E(G) \). Since Operation 4 cannot be performed on \( G \), \( |B| \geq 2 \). Moreover, by Lemma 29, \( B \subseteq V(P) \).

Let \( u_1, \ldots, u_5 \) be the vertices of \( P \) and assume that they appear in \( P \) in this order. Then, \( P \) is a dead 4-path component of \( C_0 \). So, by Statement 5 in Lemma 17, both \( u_1 \) and \( u_5 \) are leaves of \( G \). Thus, \( B \subseteq \{u_2, u_3, u_4\} \). Since \( |C| = 4 \) and \( C \) has at least three ports (by Statement 6 in Lemma 17), there are two consecutive edges in \( C \) whose endpoints are all ports of \( C \). Without loss of generality, we assume that \( v_1 \) through \( v_3 \) are ports of \( C \).

**Case 1** \( \{v_2, u_3\} \in E(G) \). In this case, since both \( v_1 \) and \( v_3 \) are ports of \( C \) and Operation 12 cannot be performed on \( C_0 \), \( \{u_2, u_3, u_4\} \cap N_G(v_1) = \{u_3\} \) and \( \{u_2, u_3, u_4\} \cap N_G(v_3) = \{u_3\} \), and in turn \( \{u_2, u_3, u_4\} \cap N_G(v_2) = \{u_3\} \) as well. Now, since \( |B| \geq 2 \), \( G \) has an edge \( \{v_4, u_j\} \) with \( j \in \{2, 4\} \). However, we can now see that Operation 12 can be performed on \( C_0 \), a contradiction.

**Case 2** \( \{v_2, u_3\} \notin E(G) \). In this case, since \( v_2 \) is a port of \( C \), \( \{v_2, u_2\} \in E(G) \) or \( \{v_2, u_4\} \in E(G) \). So, \( \{v_1, u_3\} \notin E(G) \) and \( \{v_3, u_3\} \notin E(G) \), because Operation 12 cannot be performed on \( C_0 \). Thus, \( N_G((v_1, v_2, v_3)) \cap \{u_2, u_3, u_4\} \) is either \( \{v_2, u_4\} \) or \( \{u_j\} \) for some \( j \in \{2, 4\} \). In the latter case, since \( |B| \geq 2 \), \( \{v_4, u_3\} \in E(G) \) or \( \{v_4, u_6-j\} \in E(G) \), and hence Operation 10 or 12 can be performed, a contradiction. In the former case, if \( \{v_4, u_3\} \in E(G) \), then Operation 12 can be performed on \( C_0 \), a contradiction; otherwise, \( N_G(v_4) \subseteq V(C) \cup \{u_2, u_4\} \) and in turn Operation 10 can be performed on \( G \), a contradiction. □

**Lemma 32** No 4-cycle of \( C_2 \) is adjacent to a 5-cycle of \( C_2 \) in \( G \).

**Proof** For a contradiction, assume that a 4-cycle \( C_1 \) of \( C_2 \) is adjacent to a 5-cycle \( C_2 \) of \( C_2 \) in \( G \). For each \( i \in \{1, 2\} \), let \( v_{i, 1}, \ldots, v_{i, |C_i|} \) be the vertices of \( C_i \) and assume that they appear in \( C_i \) clockwise in this order. By Statement 6 in Lemma 17, \( C_1 \) has at least three ports, and in turn three ports of \( C_1 \) appear in \( C_1 \) consecutively because \( |C_1| = 4 \). Without loss of generality, we assume that \( v_{1, 1}, v_{1, 2} \), and \( v_{1, 3} \) are ports of \( C_1 \).

By Lemma 29, \( N_G(C_1) \setminus V(C_1) \subseteq V(C_2) \). Since \( |C_1| + |C_2| - 1 = 8 < 9 \) and Operation 4 cannot be performed on \( G \), \( |X| \geq 2 \), where \( X \) is the set of vertices.
$v_{2,j} \in V(C_2)$ with $N_G(v_{2,j}) \setminus (V(C_1) \cup V(C_2)) \neq \emptyset$. We distinguish two cases as follows.

**Case 1** $X$ has two vertices adjacent in $C_2$. Without loss of generality, we assume that $\{v_{2,4}, v_{2,5}\} \subseteq X$. Then, since Operation 20 cannot be performed on $C_2$, $N_G(C_1) \setminus V(C_1) \subseteq \{v_{2,2}\}$. So, Operation 4 can be performed on $G$, a contradiction.

**Case 2** No two vertices of $X$ are adjacent in $C_2$. In this case, since $|C_2| = 5$, $|X| = 2$. Without loss of generality, we assume that $X = \{v_{2,1}, v_{2,3}\}$. Then, since Operation 20 cannot be performed on $C_2$, $N_G(C_1) \setminus V(C_1) \subseteq X$. Indeed, since Operation 4 cannot be performed on $C_2$, $N_G(C_1) \setminus V(C_1) = X$. So, $|N_G([v_{1,1}, v_{1,2}]) \setminus V(C_1)| = 1$ and $|N_G([v_{1,2}, v_{1,3}]) \setminus V(C_1)| = 1$ because Operation 10 cannot be performed on $G$. Thus, $|N_G([v_{1,1}, v_{1,2}, v_{1,3}]) \setminus V(C_1)| = 1$ because $N_G(v_{1,2}) \setminus V(C_1) \neq \emptyset$. Now, since $|N_G(C_1) \setminus V(C_1)| = 2$, $|N_G([v_{1,3}, v_{1,4}]) \setminus V(C_1)| = 2$ and in turn Operation 10 can be performed on $G$, a contradiction. □

Based on the above lemmas in this section, we are now ready to prove the next lemma:

**Lemma 33** Suppose that $C$ is a connected component of $C_2$. Then, $C$ is a 4-cycle, 5-cycle, 0-path, 4-path, or good connected component. Moreover, the following statements hold:

1. If $C$ is a 0-path, then its unique vertex $u$ satisfies that for a single tree component $C'$ of $C_2$, each $v \in N_G(u)$ is an internal vertex of $C'$, and $u$ is a leaf of $G$ if $C'$ is bad.
2. If $C$ is a 4-path component of $C_2$, then its endpoints are leaves of $G$ and each internal vertex $u$ of $C$ satisfies that each neighbor of $u$ in $G$ is a leaf of $G$, a vertex of a 5-cycle of $C_2$, or an internal vertex of a 4-path component or a good connected component of $C$.
3. If $C$ is a 4-cycle of $C_2$, then each vertex $u$ of $C$ satisfies that each neighbor of $u$ in $G$ is an internal vertex of a good connected component of $C_2$.
4. If $C$ is a 5-cycle of $C_2$, then each vertex $u$ of $C$ satisfies that each neighbor of $u$ in $G$ is an internal vertex of a 4-path component of $C_2$.
5. If $C$ is a good connected component but not a Hamiltonian path of $G$, then each leaf $u$ of $C$ satisfies that each neighbor of $u$ in $G$ is an internal vertex of $C$.

**Proof** By Lemmas 19 through 28, $C$ is a cycle of length at least 4, 0-path, 4-path, or good connected component. Indeed, $C$ cannot be a cycle of length 6 or more, because otherwise Operation 6 could be performed on $C_1$ or Operation $i$ could be performed on $C_2$ for some $i \in \{15, 16, 17\}$. We next prove the statements separately as follows.

**Statement 1** Suppose that $C$ is a 0-path. Let $u$ be the unique vertex in $C$. Since Operation 18 cannot be performed on $C_2$, $N_G(u) \subseteq V(C')$ for some connected component $C'$ of $C_2$. If $C'$ is not a connected component of $C_1$, then by Lemmas 20 through 28, $C'$ is a good connected component of $C_2$ and in turn each $v \in N_G(u)$ is an internal vertex of $C'$ (because otherwise Operation 19 could be performed on $C_2$). So, we may assume that $C'$ is also a connected component of $C_1$. Then, by Lemma 19, $C'$ is a
tree component and each $v \in N_G(u)$ is an internal vertex of $C'$. For a contradiction, assume that $C'$ is bad but $u$ is not a leaf of $G$. Since $C'$ is a bad tree component of $C_1$ with internal vertices, Lemma 19 ensures that $C'$ is a 4-path. Let $u_1, \ldots, u_5$ be the vertices of $C'$ and assume that they appear in $C'$ in this order. Since $u$ is not a leaf of $G$, Lemma 19 ensures that $N_G(u) = \{u_2, u_4\}$. Now, Operation 8 or 23 can be performed on $G$, a contradiction.

**Statement 2** Suppose that $C$ is a 4-path component of $C_2$. Then, $C$ is also a 4-path component of $C_1$, because Operation $i$ does not produce a new bad connected component in $C$ for each $i \in \{15, \ldots, 23\}$. So, by Lemma 19, each endpoint of $C$ is a leaf of $G$. Consider an arbitrary internal vertex $u$ of $C$ and an arbitrary neighbor $v$ of $u$ in $G$. Since Operation 19 cannot be performed on $C_2$, $v$ is not a leaf of a good connected component of $C_2$. So, if $v$ appears in a good connected component $C'$ of $C_2$, $v$ must be an internal vertex of $C'$. Moreover, by Lemma 31, $v$ cannot appear in a 4-cycle of $C_2$. Thus, to finish the proof, we may assume that $v$ appears in a bad tree component $C''$ of $C_2$. Now, if $v$ appears in a 0-path component of $C_2$, then $v$ is a leaf of $G$ by Statement 1 in this lemma; otherwise, $v$ cannot be an endpoint of another 4-path component of $C_2$ because each endpoint of a 4-path component of $C_2$ is a leaf of $G$.

**Statement 3** Let $C$ be a 4-cycle of $C_2$. $C$ cannot be adjacent to a 0-path component of $C_2$ in $G$, because Operation 6 cannot be performed on $C_0$ and neither Stage 2 nor Stage 3 produces a new cycle or a new 0-path component in $C$. So, by Lemmas 30 through 32, each $v \in N_G(C)$ appears in a good connected component $C'$ of $C_2$. Indeed, $v$ must be an internal vertex of $C'$, because Operation 19 cannot be performed on $C_2$.

**Statement 4** Let $C$ be a 5-cycle of $C_2$. $C$ cannot be adjacent to a 0-path component of $C_2$ in $G$, because Operation 6 cannot be performed on $C_0$ and neither Stage 2 nor Stage 3 produces a new cycle or a new 0-path component in $C$. Moreover, since neither Operation 15 nor Operation 16 can be performed on $C_2$, $C$ cannot be adjacent to a 5-cycle or a good connected component of $C_2$ in $G$. So, by Lemma 32, each $v \in N_G(C)$ appears in a 4-path component $C'$ of $C_2$. Indeed, $v$ must be an internal vertex of $C'$, because each endpoint of $C'$ is a leaf of $G$.

**Statement 5** Suppose that $C$ is a good connected component of $C_2$ but not a Hamiltonian path of $G$. Let $u$ be a leaf of $C$. Since Operation $i$ cannot be performed on $C_2$ for each $i \in \{19, 21, 22\}$, each $v \in N_G(u)$ is an internal vertex of $C$. □

Finally, in the third stage, we complete the transformation of $C$ into a spanning tree of $G$ by further modifying $C$ by performing the following steps:

1. For each cycle $C$ of $C$, first select an arbitrary edge $e = \{u, v\} \in E(G)$ such that $u \in V(C)$ and $v \in V(G) \setminus V(C)$, then delete one edge incident to $u$ in $C$, and further add $e$. (Comment Since no two cycles in $C_2$ are adjacent in $G$, $v$ appears in a tree component of $C$. Moreover, after this step, $C$ has only tree components.)

2. Arbitrarily connect the connected components of $C$ into a tree by adding some edges of $G$.

It is easy to see that for each $i \in \{15, \ldots, 23\}$, Operation $i$ can be done in $O(m)$ time. So, the second stage takes $O(nm)$ time. Since the other two stages can be easily done in $O(m)$ time, the refined Step 6 can be done $O(nm)$ time.
8 Performance analysis

Let $g_2$ (respectively, $g_3$) be the number of internal vertices in connected components of $C_2$ satisfying Condition C2 (respectively, C3), $b_2$ (respectively, $b_3$) be the total number of edges in $C_0$ whose endpoints appear in the same connected components of $C_2$ satisfying Condition C2 (respectively, C3), $c_4$ (respectively, $c_5$) be the number of 4-cycles (respectively, 5-cycles) in $C_2$, and $p_4$ be the number of 4-path components in $C_2$.

**Lemma 34** Let $T_{apx}$ be the spanning tree of $G$ outputted by the refined algorithm. Then, the following hold:

1. $w(T_{apx}) \geq 3c_4 + 4c_5 + 3p_4 + g_2 + g_3 \geq 3c_4 + 4c_5 + 3p_4 + \frac{4}{3}b_2 + b_3$.
2. $\text{opt}(G) \leq 4c_4 + 5c_5 + 4p_4 + b_2 + b_3$.
3. $\text{opt}(G) \leq 3c_4 + 4c_5 + 3p_4 + 2g_2 + 2g_3$.

**Proof** We prove the statements separately as follows.

**Statement 1** Obvious.

**Statement 2** Clear from the fact that $\text{opt}(G) \leq |E(C_0)| \leq 4c_4 + 5c_5 + 4p_4 + b_2 + b_3$.

**Statement 3** For convenience, let $T'$ be obtained from $T$ by rooting $T$ at an internal vertex, and $T''$ be obtained from $T'$ by removing those edges $(u, v)$ such that some 4-cycle of $C_2$ contains both $u$ and $v$. Further let $I'$ (respectively, $I''$) be the set of vertices in $T'$ (respectively, $T''$) that have at least one child in $T'$ (respectively, $T''$). Also let $J = I' \setminus I''$. Clearly, $w(T) = |I'|$. Moreover, for each 4-cycle $C$ of $C_2$, $T'$ can contain at most three edges between the vertices of $C$. So, $|J| \leq 3c_4$. Furthermore, Lemma 33 ensures that each vertex of $I'$ other than

- the vertices in $J$,
- the $3p_4$ internal vertices of 4-path components of $C_2$,
- the $5c_5$ vertices of 5-cycles, and
- the $g_2 + g_3$ internal vertices of good connected components of $C_2$

must have a child in $T''$ that is an internal vertex of a good connected component of $C_2$. So, $w(T) = |I'| \leq 3c_4 + (5c_5 + 3p_4 + g_2 + g_3) + (g_2 + g_3)$. \hfill \Box

**Theorem 2** The algorithm achieves an approximation ratio of $\frac{13}{17}$ and runs in $O(n^2m^2) + t(2n, 2m)$ time.

**Proof** Let $T_{apx}$ be as in Lemma 34, and $r = w(T_{apx})/\text{opt}(G)$. By Lemma 34, $r \geq \max\{r_1, r_2\}$, where $r_1 = \frac{3c_4 + 4c_5 + 3p_4 + g_2 + g_3}{4c_4 + 5c_5 + 4p_4 + b_2 + b_3}$ and $r_2 = \frac{3c_4 + 4c_5 + 3p_4 + g_2 + g_3}{4c_4 + 5c_5 + 3p_4 + 2g_2 + 2g_3}$. Note that $r_1 \geq \min\left\{\frac{4}{3}, r'_1\right\}$ and $r_2 \geq \min\left\{\frac{4}{3}, r'_2\right\}$, where $r'_1 = \frac{3c_4 + 3p_4 + g_2 + g_3}{4c_4 + 4p_4 + b_2 + b_3}$ and $r'_2 = \frac{3c_4 + 3p_4 + g_2 + g_3}{4c_4 + 3p_4 + 2g_2 + 2g_3}$. So, it suffices to show that $\max\{r'_1, r'_2\} \geq \frac{13}{17}$. This is done if $r'_1 \geq \frac{13}{17}$.

Thus, we assume that $r'_1 < \frac{13}{17}$. Then, $c_4 + p_4 > 17g_2 + 17g_3 - 13b_2 - 13b_3$. Hence,

$$r'_2 > \frac{52g_2 + 52g_3 - 39b_2 - 39b_3}{53g_2 + 53g_3 - 39b_2 - 39b_3} \geq \min\left\{\frac{52g_2 - 39b_2}{53g_2 - 39b_2}, \frac{52g_3 - 39b_3}{53g_3 - 39b_3}\right\}.$$ \hfill \Box

Moreover, since $g_3 \geq b_3$, $\frac{52g_3 - 39b_3}{53g_3 - 39b_3} \geq \frac{13}{17}$. Therefore, $r'_2 > \frac{13}{17}$. The running is clearly as claimed.

Recall that $t(n, m) = O(n^2m^2)$ (Hartvigsen 1984). So, the algorithm takes $O(n^2m^2)$ time.
Acknowledgements  Zhi-Zhong Chen was supported in part by the Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan, under Grant No. 24500023. Lusheng Wang was fully supported by NSFC (61373048) and GRF (CityU 123013) grants.

References

Binkele-Raible D, Fernau H, Gaspers S, Liedloff M (2013) Exact and parameterized algorithms for max internal spanning tree. Algorithmica 65(1):95–128
Chen ZZ, Lin G, Wang L, Chen Y (2016) Approximation algorithms for the maximum weight internal spanning tree problem. CoRR. arxiv:abs/1608.03299
Coben N, Fomin FV, Gutin G, Kim EJ, Saurabh S, Yeo A (2010) Algorithm for finding k-vertex out-trees and its application to k-internal out-branching problem. JCSS 76:650–662
Fomin FV, Lokshtanov D, Grandoni F, Saurabh S (2012) Sharp separation and applications to exact and parameterized algorithms. Algorithmica 63:692–706
Fomin FV, Gaspers S, Saurabh S, Thomasse S (2013) A linear vertex kernel for maximum internal spanning tree. JCSS 79:1–6
Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W.H. Freeman, New York
Hartvigsen D (1984) Extensions of matching theory. Ph.D. Thesis, Carnegie-Mellon University
Knauer M, Spoerhase J (2009) Better approximation algorithms for the maximum internal spanning tree problem. In: Proceedings of WADS, LNCS, vol 5664, pp 459–470
Li X, Zhu D (2014) Approximating the maximum internal spanning tree problem via a maximum path-cycle cover. In: Proceedings of ISAAC, LNCS, vol 8889, pp 467–478. CoRR. arxiv:abs/1409.3700
Li W, Chen J, Wang J (2014) Deeper local search for better approximation on maximum internal spanning tree. In: Proceedings of ESA, LNCS, vol 8737, pp 642–653
Li W, Wang J, Chen J, Cao Y (2015) A 2k-vertex kernel for maximum internal spanning tree. In: Proceedings of WADS, LNCS, vol 9214, pp 495–505
Li X, Feng H, Jiang H, Zhu B (2016) A polynomial time algorithm for finding a spanning tree with maximum number of internal vertices on interval graphs. In: Proceedings of FAW, LNCS, vol 9711, pp 92–101
Prieto E (2005) Systematic kernelization in FPT algorithm design. Ph.D. Thesis, The University of Newcastle, Australia
Prieto E, Sliper C (2003) Either/or: using vertex cover structure in designing FPT-algorithms—the case of k-internal spanning tree. In: Proceedings of WADS, LNCS, vol 2748, pp 474–483
Prieto E, Sloper C (2005) Reducing to independent set structure—the case of k-internal spanning tree. Nord J Comput 12:308–318
Salamon G (2009a) Approximating the maximum internal spanning tree problem. Theor Comput Sci 410(50):5273–5284
Salamon G (2009b) Degree-based spanning tree optimization. Ph.D. thesis, Budapest University of Technology and Economics, Hungary
Salamon G, Wiener G (2008) On finding spanning trees with few leaves. Inf Process Lett 105(5):164–169