A Polynomial Time Algorithm for Steiner Tree When Terminals Avoid a Rooted K_4-Minor

Carla Groenland
Delft Institute of Applied Mathematics, The Netherlands

Jesper Nederlof
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Tomohiro Koana
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Abstract

We study a special case of the Steiner Tree problem in which the input graph does not have a minor model of a complete graph on 4 vertices for which all branch sets contain a terminal. We show that this problem can be solved in $O(n^4)$ time, where n denotes the number of vertices in the input graph. This generalizes a seminal paper by Erickson et al. [Math. Oper. Res., 1987] that solves Steiner tree on planar graphs with all terminals on one face in polynomial time.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Steiner tree, rooted minor

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.12

Related Version *ArXiv Version*: https://arxiv.org/abs/2410.06793

Funding The work of both JN and TK has been supported by the project CRACKNP that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 853234).

1 Introduction

Planar graphs are well-studied algorithmically. For example, starting with the work of Baker [1], many efficient approximation schemes for NP-complete problems on planar graphs have been designed. Within parameterized complexity, widely applicable tools such as bi-dimensionality [6] helped to grasp a firm understanding of the "square-root phenomenon": Many problems can be solved in $2^{O(\sqrt{n})}$ time, or even in $2^{O(\sqrt{k})} \cdot n^{O(1)}$ time, where n denotes the number of vertices of the input graph and k denotes the size of the sought solution. Actually most of these algorithms are also shown to extend to superclasses[1] of planar graphs, such as bounded genus or even minor free graphs (as for example showed in [6]).

One problem that is very well-studied on planar graphs is Steiner Tree. In the Steiner Tree problem we are given a weighted graph $G = (V, E)$ on n vertices with edge weights $w_e \in \mathbb{R}_{\geq 0}$ for $e \in E$ and a set of vertices $T = \{t_1, \ldots, t_k\} \subseteq V$ called *terminals*, and we are tasked with finding an edge set S minimizing $w(S) = \sum_{e \in S} w(e)$ such that any pair of terminals t, t' are connected in the subgraph (V, S). For example, an efficient approximation scheme was given in [4], and a lower bound excluding $2^{o(k)}$ time algorithms on planar graphs under the Exponential Time Hypothesis was given in [17]. Interestingly, the latter result shows that planarity alone is not too helpful to solve Steiner Tree quickly, because it implies that the classic $3^k n^{O(1)}$ dynamic programming algorithm for general Steiner Tree [8] cannot be significantly improved.

[1] Wagner's theorem states that a graph is planar unless it contains a complete graph on 5 vertices (K_5) or complete bipartite graph with two blocks of 3 vertices ($K_{3,3}$) as a minor.

A seminal paper by Erickson et al. [9] shows that Steiner Tree on planar graphs is in P if all terminals lie on one face in a planar embedding of G. The study of the setting in which terminals lie in a few number of faces dates back all the way to the work of Ford and Fulkerson [11], and has been the subject of many classic works (such as the Okamura-Seymour Theorem [21, Chapter 74] or [18]). The algorithm [9] is used as subroutine in several other papers such as the aforementioned approximation scheme [4], but also preprocessing algorithms [19].

Often the algorithms that exploit planarity or minor-freeness need to combine *graph theoretic* techniques (such as a grid minor theorem in the case of bi-dimensionality) with *algorithmic* perspective (such as divide and conquer or dynamic programming over tree decompositions). For the Steiner Tree problem, and especially the algorithm from [9], the graph theoretic techiques employed are intrinsically *geometric*. And indeed, many extensions of the algorithm, such as the one in which the terminals can be covered the k outer faces of the graph [2], or the setting in which the terminals can be covered by a "path-convex region" studied in [20], all have a strong geometric flavor.

With this in mind, it is natural to ask whether there is an extension of the algorithm of [9] to minor free graphs.

Rooted Minors

We study a setting with *rooted* minors. A graph H is a *minor* of a graph G if it can be constructed from G by removing vertices or edges, or contracting edges. In a more general setting, G has a set of *roots* $R \subseteq V(G)$ and there is a mapping π that prescribes for each $v \in V(H)$ the set of roots $\pi(v) \subseteq R$. Then *a rooted minor* is a minor that contracts r into v, for every $v \in V(H)$ and $r \in R$ such that $r \in \pi(v)$. Rooted minors play a central in Robertson and Seymour's graph minor theory, and directly generalize the Disjoint Paths problem. Recently it was shown [16] that rooted minors can be detected in $(m+n)^{1+o(1)}$ time, for fixed $|H|$ and $|R|$, improving over the quadratic time algorithm from [15]. A number of recent papers have studied rooted minors in their own right [3, 12–14, 22, 23]. In particular, Fabila-Monroy and Wood [10] gave a characterization of when a graph contains a K_4-minor rooted at four given vertices. Recently, links between rooted minors and "rooted" variants of treewidth, pathwidth and treedepth were given in [12].

Motivated by the missing extension of the algorithm of [9] to a (non-geometric) setting of excluded minors, we propose studying the complexity of instances without minors rooted at the terminals.

Our Result

We will be interested in the closely related (but slightly different) setting of *R-rooted K_4-minor*: In a graph $G = (V, E)$ with vertices $R \subseteq V$ (referred to as *roots*), an *R-rooted K_4-minor* is a collection of disjoint vertex sets $S_1, S_2, S_3, S_4 \subseteq V$ such that $G[S_i]$ is connected and $S_i \cap R \neq \emptyset$ for all $i \in \{1, \ldots, 4\}$, and such that there is an edge from S_i to S_j for each $i \neq j$.

▶ **Theorem 1.** *Steiner Tree without K_4 minor rooted at terminals can be solved in $O(n^4)$ time.*

This generalizes the result from [9]: It is easily seen that a planar graph has no K_4-minor rooted at four vertices on the same face. On the other hand, it is easy to come up with example instances that have no K_4-minor and are not planar (see Figure 1 for such two such examples).

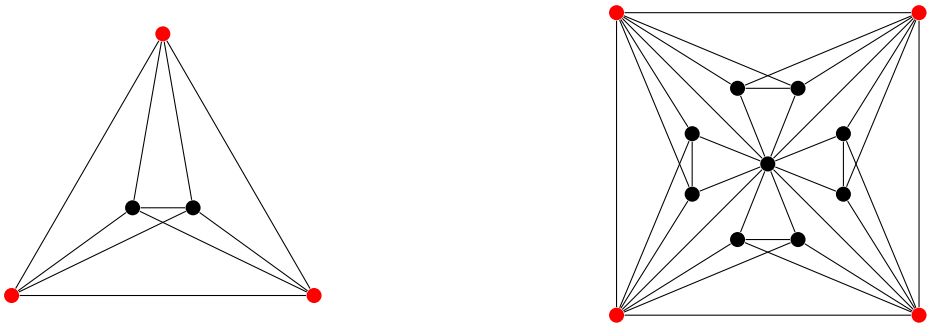

Figure 1 Terminals are depicted in red. The left figure has 3 terminals and hence no rooted K_4 minor, but it is a K_5 and hence not planar. The right figure has no rooted K_4 minor (since the middle vertex and two non-adjacent terminals separate the remaining terminals), and has many K_5 subgraphs and hence is not planar.

We hope that our result paves the road for additional polynomial time algorithms (solving non-geometrically) restricted versions of Steiner Tree. In particular, it would be interesting to see whether Theorem 1 can be extended to a polynomial time algorithm for a richer set of rooted $\mathcal{F}$-minor free instances (e.g., instances that do not contain any member of $\mathcal{F}$ as rooted minor). Note however that such a strengthening with $\mathcal{F} = \{K_5\}$ would imply P=NP since Steiner Tree on planar graphs is NP-hard.

Our Approach

It is not too difficult to show from ideas given in [10] that if there is no R-rooted K_4-minor in a 3-connected graph, then there is a cycle C passing through all vertices in R. We need a slightly stronger variant of this result, where the graph is not quite 3-connected but the only 2-cuts allowed are those which isolate a single vertex that is a terminal (see Lemma 4). Our algorithm will recurse on 2-cuts until it can apply this lemma, and then performs dynamic programming in a similar fashion to the Dreyfus-Wagner algorithm [8] restricted to subsets of terminals that form a contiguous segment of the cycle C (in a fashion that is analogous to the algorithm for [9], although the virtual edges considerably increase the technical difficulty of our proof). Since the number of such segments is at most quadratic in n, this gives a polynomial time algorithm.

We stress that the approach of [9] does not work directly: The approach of [9] crucially relies on the fact that if one decomposes an optimal Steiner Tree S into two trees $S_1 \cup S_2$ where S_1, S_2 are the connected components of $S \setminus e$ for some $e \in S$, then the set R_1 (and R_2) of terminals covered by S_1 (and S_2) is a contiguously visited along the cycle enclosing the face that contains all terminals. In our setting, there is a priori no guarantee how the set R_1 will look, and therefore such a decomposition approach will not work. To overcome this, we also decompose the optimal tree along 2-cuts via recursion and processing outcomes of recursive calls with virtual edges. Even though the virtual edges lead to some technical challenges, this allows us to make the idea of [9] work in 3-connected graphs.

2 Preliminaries

For $n \in \mathbb{N}$, we define $[n]$ as the set $\{1, \ldots, n\}$. In this work, we assume that all graphs do not have self-loops, but we allow the graph to have parallel edges. We will say that a graph is *simple* if there is no parallel edge. The vertex set and edge set of a graph G are denoted by $V(G)$ and $E(G)$, respectively.

For two vertices u and v, let $\mathrm{dist}(u,v)$ be the length of a shortest path between u and v. Let $X \subseteq V(G)$ be a subset of vertices. We use $G[X]$ to denote the subgraph induced by X and $G - X := G[V(G) \setminus X]$ to represent the graph obtained by removing the vertices in X. For these notations, if X is a singleton $\{x\}$, we may write x instead of $\{x\}$. Moreover, X is called a *cut* if deleting X increases the number of connected components. In particular, x is called a *cut vertex* if $\{x\}$ is a cut. For an edge set $F \subseteq E$, let $V(F)$ denote the set of vertices *covered* by F, i.e., $V(F) = \{u \mid u \in e \in F\}$.

Avoiding a (rooted) minor is preserved under deletion of vertices, deletion of edges and contraction of edges.

▶ **Observation 2.** *If G does not contain an R-rooted K_4-minor and H is a minor of G, then H does not contain an $(V(H) \cap R)$-rooted K_4-minor.*

Our structural analysis will also crucially build on the following simple lemma.

▶ **Lemma 3** (Lemma 7 in [10])**.** *Suppose that a graph G has a cycle containing vertices v_1, v_2, v_3, v_4 in that order. Suppose moreover that there are two disjoint paths P_1 and P_2 where P_1 is from v_1 to v_3, and P_2 is from v_2 to v_4. Then G has a $\{v_1, v_2, v_3, v_4\}$-rooted K_4-minor.*

3 Finding a cycle passing through terminals and virtual edges

In this section, we prove a structural lemma that our algorithm needs. This builds on ideas from [10], but we require additional analysis due to the presence of "virtual edges".

▶ **Lemma 4.** *Let $G = (V, E)$ be a 3-connected graph and let $R \subseteq V$ be a set of roots with $|R| \geq 3$. Let $E' \subseteq E$ and let G' be obtained by subdividing each edge in E' once. Let S denote the set of subdivision vertices added with this operation.*

If G' has no $(R \cup S)$-rooted K_4-minor, then G' contains a cycle containing all vertices in $R \cup S$. Furthermore, this cycle can be found in $nm^{1+o(1)}$ time for n the number of vertices and m the number of edges of G'.

Proof. We first show that if there is no R-rooted K_4-minor in G and G is 3-connected, then we can find a cycle C in G that passes through all vertices of R.

We start with C being any cycle. Suppose there is a vertex $r \in R$ that is not on C. Since G is 3-connected, by Menger's theorem (see [7, Theorem 3.3.1], applied with sets C and $N(v)$), there exist three paths P_1, P_2, P_3 from r to C, where the paths mutually only intersect in r and each path only intersects C in their other endpoint. We can find these paths in $m^{1+o(1)}$ time with the max flow algorithm from e.g. [5]. Let $v_1, v_2, v_3 \in C$ denote these endpoints. If there are three disjoint arcs on the cycle that each contains a root from R and one of $\{v_1, v_2, v_3\}$, then G contains a rooted K_4 minor (see Figure 2).

But if such arcs do not exist, then after renumbering we may assume that there is no root vertex between v_1 and v_2 on C. The cycle C' which goes from v_1 to v_2 via the path contained in C containing v_3, and then back to v_1 via r via P_1 and P_3, forms a cycle containing $(R \cap V(C)) \cup \{r\}$. Hence we can reiterate with this cycle until C contains R.

This means there is also a cycle C in G' containing all vertices in R: whenever an edge in E' is used, it is possible to pass through the subdivision vertex instead. We next modify the cycle C to be a cycle containing all vertices in R and all vertices in S.

Let $uv \in E'$ be the edge whose subdivision resulted in s. We first ensure that C contains both u and v (and R and all vertices in S already contained in it).

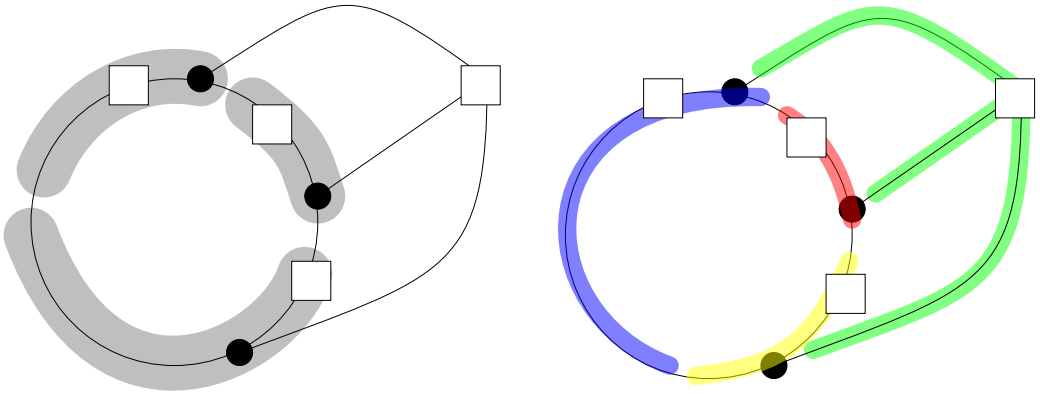

Figure 2 If there are three vertex-disjoint paths from a root to disjoint arcs of the cycle containing roots (left), we obtain a rooted K_4-minor (right). The roots are depicted by boxes.

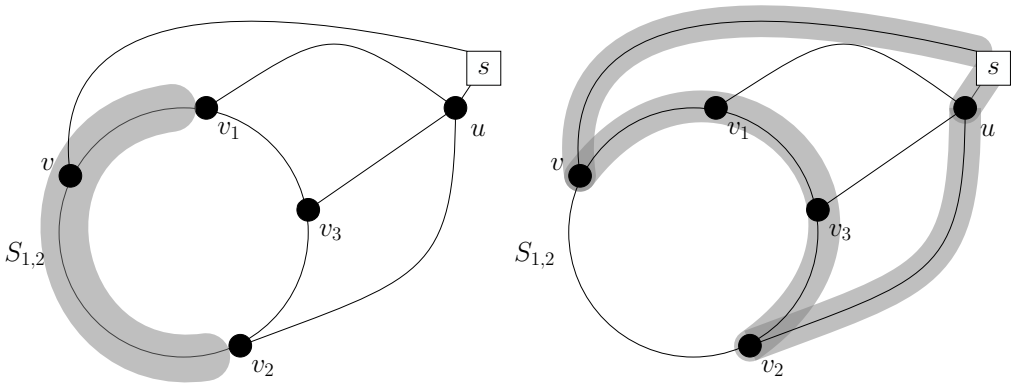

Figure 3 If the segment $S_{1,2}$ contains v and no roots of C (left), then there is a cycle through v, u, s and all roots of C.

Suppose that u is not in C. As before, since G is 3-connected and $u \in V(G)$, there must be three paths from u to C, vertex-disjoint except for their endpoint u, meeting C in distinct endpoints v_1, v_2, v_3. Again, we can find these paths in $m^{1+o(1)}$ time with [5]. We consider the same two cases as before.

- *Case 1: there are three disjoint arcs on C, each of which contains a vertex of $R \cup S$ and one vertex from $\{v_1, v_2, v_3\}$.* After contracting the edge between s and u, we obtain a rooted K_4-minor in the same way as in the 3-connected case (see Figure 2). Hence this case does not happen by assumption.
- *Case 2: the segment $S_{1,2}$ between v_1 and v_2 (excluding v_1, v_2 and v_3) contains no vertices from $S \cup R$.* Then there is a cycle C' containing $V(C) \cap (S \cup R)$ and u. Note that C' no longer contains the vertices from $S_{1,2}$, so we could potentially lose v if $v \in S_{1,2}$. But if $v \in S_{1,2}$, then there is a cycle C'' that goes through $V(C) \cap (S \cup R) \cup \{s\}$ (see Figure 3).

Either way we ensured that the cycle contains u without affecting whether it contains v. After renumbering if needed, we are always either in Case 1 or Case 2. Repeating the argument above for v if needed, we can find a cycle passing through $(R \cup S) \cap V(C) \cup \{u, v\}$.

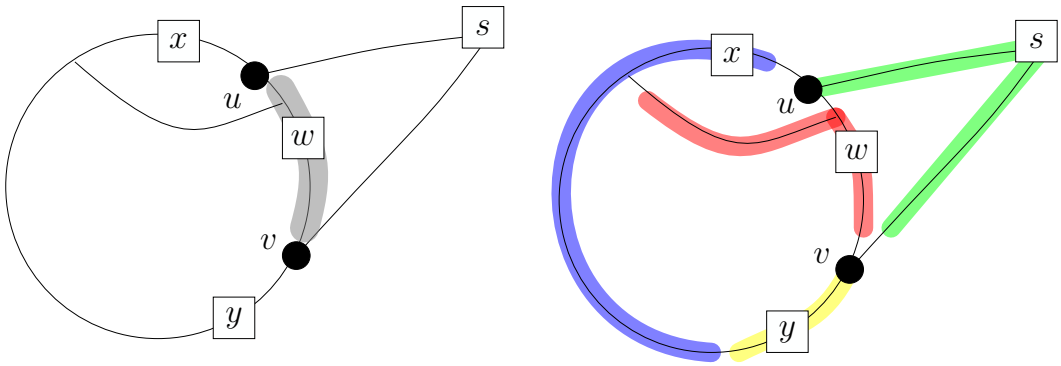

Figure 4 The segment S_{uv} contains vertices between u and v, including w and excluding u and v itself (left). If there is a path from S_{uv} to x or the segment between x and y, then there is a rooted K_4-minor (right), where v is included with y and u with s.

So we will assume C passes through u and v. Now we claim that one of the two arcs of C between $u, v \in C$ does not contain any vertices from $S \cup R$, and hence we may replace this by the path via s to get a cycle C that additionally passes through s.

Suppose this is not the case. Recall we assumed in our lemma statement that there are at least three vertices in R, and already established that C contains all vertices of R. So we can find w, x, y in $R \cup S$ such that C passes in order through u, w, v, y, x, u (with some vertices in between those, possibly). We choose x, y such that there are no vertices from $R \cup S$ between v and y and between x and u. (See also Figure 4.)

Let S_{uv} be the maximum segment of C between u and v which includes w but excludes u and v. That is, if $u, a_1, \ldots, a_\ell, w, b_1, \ldots, b_k, v$ are the vertices of C between u and v containing w, then

$$S_{uv} = \{a_1, \ldots, a_\ell, w, b_1, \ldots, b_k\}.$$

If there is a path intersecting C only in its endpoints from a vertex in S_{uv} to x or the segment of C in between x, y that excludes S_{uv}, then we have an $(S \cup R)$-rooted K_4-minor in G', as shown in Figure 4.

Similarly, if there is a path from S_{uv} to y, a vertex in the segment between x and u (not including u) or the segment between y and v (not including v), then there is an $(S \cup R)$-rooted K_4-minor in G', as depicted in Figure 5.

Since w and x are not the subdivision vertex of u and v, S_{uv} and $V(C) \setminus (\{u, v\} \cup S_{uv})$ both contain at least one vertex from G. Therefore, there is a path in $G \setminus \{u, v\}$ from $S_{uv} \cap V(G)$ to $(C \setminus S_{uv}) \cap V(G)$, since G is 3-connected. This means that one of the paths mentioned above exists, contradicting the fact that G' has no $(S \cup R)$-rooted K_4-minor. This contradiction shows that C must contain all vertices of S, as desired.

Hence, one of the two arcs of C between $u, v \in C$ does not contain any vertices from $S \cup R$, and we may replace this by the path via s to get a cycle C that additionally passes through s. Reiterating this argument at most n times gives the cycle passing through $R \cup S$. Since each iteration takes at most $m^{1+o(1)}$ time, the lemma follows. ◀

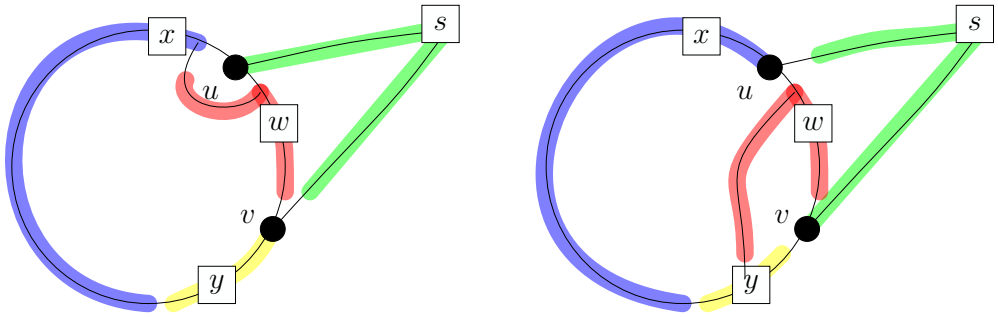

Figure 5 If there is a path from S_{uv} to the segment between x and u (not including u), then there is a rooted K_4-minor (left). If there is a path from S_{uv} to y, then there is a rooted K_4-minor (right).

4 Description of algorithm

This section is organized as follows. In Section 4.1, we define an auxiliary problem called Virtual Edge Steiner Tree. In Section 4.2 we present a dynamic programming algorithm for Virtual Edge Steiner Tree for the case that all roots lie on a cycle. This cycle will be provided to us by Lemma 4. Section 4.3 discusses preprocessing steps. Finally, in Section 4.4, we describe our algorithm for Virtual Edge Steiner Tree.

4.1 Virtual edges

We first slightly generalize the Steiner Tree problem to a problem that we call Virtual Edge Steiner Tree in order to facilitate a recursive approach.

This problem is defined as follows. The input to Virtual Edge Steiner Tree is

- a graph $G = (V, E)$ with weights w_e for $e \in E$,
- a set of terminals $T = \{t_1, \ldots, t_k\} \subseteq V$,
- a set of "virtual edges" E^* with weights $w_{e^*} : \{u, v, \mathsf{d}, \mathsf{c}\} \to \mathbb{R}_{\geq 0}$ for $e^* = \{u, v\} \in E^*$.

We will assume that $E \cap E^* = \emptyset$.

A *solution* to the Virtual Edge Steiner Tree problem is a (virtual) edge set $S \subseteq E \cup E^*$ such that $T \subseteq V(S)$ and $V(S) \cap V(e^*) \neq \emptyset$ for each virtual edge $e^* \in E$. Since the edge weights are non-negative we can assume S is a tree. For a (virtual) edge set S, the *cost* $c_{e^*}(S)$ of a virtual edge $e^* = \{u, v\} \in E^*$ with respect to S is given by

- $w_{e^*}(u)$ if $u \in V(S)$ and $v \notin V(S)$, representing the cost when u is included in the solution and v is not,
- $w_{e^*}(v)$ if $v \in V(S)$ and $u \notin V(S)$, representing the cost when v is included in the solution and u is not,
- $w_{e^*}(\mathsf{c})$ if $e \in S$, representing the cost when u, v are both included in the solution and connected "via the virtual edge", that is, in the part of the graph we "forgot about",
- $w_{e^*}(\mathsf{d})$ if $u, v \in V(S)$ and $e \notin S$, representing the cost when u, v are both included in the solution but are not connected "via the virtual edge".

The *total cost* of a solution S is defined as

$$c(S) = \sum_{e \in S} w_e + \sum_{e^* \in E^*} c_{e^*}(S). \tag{1}$$

The purpose of virtual edges with cost functions is to enable our algorithm to handle 2-cuts $\{u, v\}$ effectively. When the algorithm identifies such a cut, it recursively solves the four subproblems for the smaller side first, encoding the costs in $w_{\{u,v\}}$. It is crucial to ensure that the solutions on both sides of the cut agree on the inclusion of vertices u and v in the final solution. Additionally, if both vertices are included, the algorithm must determine which side will contain the path connecting them.

In an instance of Virtual Edge Steiner Tree, we will assume that there is no terminal incident with a virtual edge. If there is a terminal t and a virtual edge tt', then we assign the virtual edge weight $w_{tt'}(t') = \infty$, and remove t from the terminal set.

It will be convenient for the description of the dynamic program to view a root as either a singleton set consisting of a vertex (if it is a terminal), or an edge (if it is a virtual edge). Therefore, for an instance $(G, w, T, E^*, \{w_{e^*}\}_{e^* \in E^*})$, we define the set of *roots* as $R = \{\{t\} : t \in T\} \cup E^*$. We say the instance has no rooted K_4-minor if the graph G^* has no R^*-rooted K_4-minor, where G^* is the graph obtained from the graph $(V, E \cup E^*)$ by subdividing each edge in E^* once, and R^* is the union of T and the subdivision vertices of G^*.

Note that Virtual Edge Steiner Tree can be reduced to Steiner Tree as follows:

▶ **Observation 5.** *There is a reduction from an instance of Virtual Edge Steiner Tree with k terminals and ℓ virtual edges to 4^ℓ instances of Steiner Tree with at most $k + 2\ell$ terminals.*

Proof. Recall that there are four cases for each virtual edge: (i) u is covered but v is not, (ii) v is covered but u is not, (iii) both u and v are covered and the edge uv is part of the solution, and (iv) u and v are covered and the edge uv is not part of the solution. For each virtual edge, we delete it from the graph, and do one of the following. For case (i), add u as a terminal and delete v. For case (ii), add v as a terminal and delete u. For case (iii), contract u and v into a single vertex and add it as a terminal. For case (iv), add both u and v as terminals. This results in a Steiner Tree instance with at most $k + 2\ell$ terminals. The minimum cost of solutions among these instances plus the virtual edge weights is the minimum cost of the original instance of Virtual Edge Steiner Tree. ◀

4.2 Dynamic programming when roots lie on a cycle

In this subsection, we present a polynomial-time algorithm for the Virtual Edge Steiner Tree problem when all roots lie on a cycle.

▶ **Lemma 6.** *An instance of Virtual Edge Steiner Tree on an n-vertex graph without rooted K_4-minor that has a cycle passing through all roots, can be solved in $O(n^4)$ time.*

Proof. We may also assume that there are at least five roots, since otherwise the problem can be efficiently solved via Observation 5. Let $R = \{r_1, \ldots, r_k\}$ denote the set of roots. We renumber the indices so that $r_1, \ldots, r_k$ appear on C in the order of their indices. Recall that no terminal is incident with a virtual edge, and since the virtual edges lie on C, each non-terminal is incident with at most two virtual edges. Indices will be considered modulo k (identifying r_k with r_0). We use the shorthand $R[a, b]$ to denote the set of roots $\{r_c \in R : c \in [a, b]\}$. In particular, $R[a, b] = \{r_a, \ldots, r_k, r_1, \ldots, r_b\}$ for $a > b$.

We say a tree $\mathcal{T} = (V(\mathcal{T}), E(\mathcal{T}))$ is a *partial interval solution* if $V(\mathcal{T}) \subseteq V(G)$, $E(\mathcal{T}) \subseteq E \cup E^*$ and the set $R(\mathcal{T})$ of roots $r \in R$ for which $r \cap V(\mathcal{T}) \neq \emptyset$ forms an interval $R[a, b]$ for some $a, b \in [k]$.

We determine the status of virtual edges incident with the partial interval solution in the same way as we do for a solution, and we also assign a cost in a way similar to (1), but we now ignore virtual edges that are not incident with S

$$c(\mathcal{T}) = \sum_{e \in E(\mathcal{T}) \cap E} w_e + \sum_{e^* \in E(\mathcal{T}) \cap E^*} c_{e^*}(E(\mathcal{T})).$$

Note that each solution $S \subseteq E \cup E^*$ gives rise to a partial interval solution $\mathcal{T} = (V(E) \cup V(E^*), E \cup E^*)$, since $R(\mathcal{T}) = [k]$ is always an interval.

We say a partial interval solution $\mathcal{T}$ is a *minimal solution* if $R(\mathcal{T}) = R$ (so it contains all terminals and contains at least one endpoint per virtual edge) and it is minimal in the sense that there is no strict subtree which also has this property. The last assumption is needed for technical reasons since the weights could also be zero.

We compute a dynamic programming table DP with entries $\mathsf{DP}[a, b, v, s_a, s_b]$, where $a, b \in [k]$, $v \in V(G) \setminus T$, $s_a \in r_a \cup \{\mathsf{d}, \mathsf{c}\}$ and $s_b \in r_b \cup \{\mathsf{d}, \mathsf{c}\}$. If r_a is a terminal, then the table may ignore s_a: we will store the same value irrespective s_a. The same applies to s_b.

The table entry $\mathsf{DP}[a, b, v, s_a, s_b]$ represents the minimum cost of a partial interval solution $\mathcal{T}$ with $R(\mathcal{T}) \supseteq R[a, b]$ and $v \in V(\mathcal{T})$, where the status of r_a and r_b are indicated by s_a and s_b, respectively. In fact, the minimum will not go over all partial interval solutions, but only those that can be built in a specific manner as defined next. This means the lower bound is always true.

We define a collection $\mathcal{B}$ of partial interval solutions via the following set of rules.

R1. Every partial interval solution $(\{v\}, \emptyset)$ consisting of a single vertex with $v \in V(G)$ is in $\mathcal{B}$.

R2. Every partial interval solution $\mathcal{T}$ obtained from a partial interval solution $\mathcal{T}' \in \mathcal{B}$ by adding a new vertex $v \in V(G)$ not incident to any roots with an edge in E is in $\mathcal{B}$.

R3. Every partial interval solution $\mathcal{T}$ obtained from two partial interval solutions $\mathcal{T}_1, \mathcal{T}_2 \in \mathcal{B}$ by "gluing on v" is in $\mathcal{B}$ when $V(\mathcal{T}_1) \cap V(\mathcal{T}_2) = \{v\}$ and all points in the intersection of the intervals $R(\mathcal{T}_1)$ and $R(\mathcal{T}_2)$ are endpoints of both intervals.

Note that by the definition of partial interval solution, $R(\mathcal{T})$ is an interval $R[a, b]$ for each $\mathcal{T} \in \mathcal{B}$. The following claim shows our dynamic program may restrict itself to partial interval solutions.

▷ Claim 7. Let $\mathcal{T}$ be a minimal solution. Then $\mathcal{T} \in \mathcal{B}$.

Proof. We want to root our tree $\mathcal{T}$ in a well-chosen vertex v and eventually prove the claim by induction. For $w \in V(\mathcal{T})$, let $\mathcal{T}_w$ denote the subtree of $\mathcal{T}$ containing all descendants of w. The choice of v needs to ensure that $R(\mathcal{T}_w) \neq [k]$ for all children w of v. By minimality, $R(\mathcal{T}_w) \neq \emptyset$ for all children w of v.

If there is at least one terminal r_a, then we can choose $v = r_a$, since r_a will then be missing from $R(\mathcal{T}_w)$ for all $w \neq v$. We show in the remainder of this paragraph that such a vertex v can also be chosen when no terminal exists. If there is a virtual edge incident to a single vertex u, we can choose $v = u$ similar to the terminal case. Moreover, if there is a vertex u incident with two virtual edges, then we may choose $v = u$. Otherwise, all roots come from virtual edges that have both endpoints present in $\mathcal{T}$ and each vertex is incident with at most one edge. We give each vertex of G incident with a virtual edge weight 1. Since $\mathcal{T}$ is a tree, we can root it in a "balanced separator": a vertex v such that the total weight in each component of $\mathcal{T} - v$ is at most half the total weight of $\mathcal{T}$. The only way $R(\mathcal{T}_w)$ can then be $[k]$ for w a child of v, is when all roots are coming from edges incident with one vertex in $\mathcal{T}_w$ and one outside. Since there are at least five roots, we can choose roots r_a, r_b, r_c, r_d

consecutive on the cycle, and connect r_a to r_c using only internal vertices from $V(\mathcal{T}_w)$ and r_b to r_d using only internal vertices not in $V(\mathcal{T}_w)$, finding a rooted K_4-minor with Lemma 3, a contradiction.

In this paragraph, we show that $R(\mathcal{T}_w)$ is an interval. We are done if $|R(\mathcal{T}_w)| \geq k-1$. So we assume that $|R(\mathcal{T}_w)| \leq k-2$ and $w \neq v$. Suppose towards a contradiction that $R(\mathcal{T}_w)$ is not an interval. This means there exist $a < b < c < d$ in $[k]$ such that $r_a, r_c \in R(\mathcal{T}_w)$ and $r_b, r_d \notin R(\mathcal{T}_w)$ (or $b < c < d < a$, but this case is analogous). This means there is a path between (the roots of) r_a and r_c via vertices in $V(\mathcal{T}_w)$. Moreover, r_b and r_d are not in $R(\mathcal{T}_w)$, and so they are either part of the tree $\mathcal{T}'$ obtained from $\mathcal{T}$ by removing $\mathcal{T}_w$, or incident to a vertex in this tree. This provides a path between (the roots of) r_b and r_d using vertices not in $V(\mathcal{T}_w) \cup R(\mathcal{T}_w)$. Thus, by Lemma 3, we obtain an $\{r_a, r_b, r_c, r_d\}$-rooted K_4-minor, a contradiction.

In this paragraph we show that $\mathcal{S}_w = (\{w\}, \emptyset) \in \mathcal{B}$ for each $w \in V(\mathcal{T})$. For this, we need to show that $R(\mathcal{T})$ forms an interval. Recall that no terminal is incident with a virtual edge, and since the virtual edges lie on C, each non-terminal is incident with at most two virtual edges which are then also consecutive on the cycle. This means $|\mathcal{S}_w| \leq 2$ and that $R(\mathcal{S}_w)$ is an interval. Since $\mathcal{S}_w$ consists of a single vertex, it is now added to $\mathcal{B}$ in **R1**.

The remainder of the proof shows that $\mathcal{T}_w \in \mathcal{B}$ for each $w \in V(\mathcal{T})$ by induction on $|V(\mathcal{T}_w)|$. We already proved (a stronger variant of) the case when the size is 1 above. Suppose $\mathcal{T}_w \in \mathcal{B}$ has been shown for all w with $|V(\mathcal{T}_w)| \leq \ell$ for some $\ell \geq 1$ and now assume $|V(\mathcal{T}_w)| = \ell + 1$.

We first show a property that we need in both cases considered below. Let w' be a child of w and suppose that $r_x \in R(\mathcal{S}_w) \cap R(\mathcal{T}_{w'})$. Then r_x must correspond to a virtual edge ww_1 with $w_1 \in V(\mathcal{T}_{w'})$. We show that r_x is an endpoint of the interval $R(\mathcal{T}_{w'})$. Recall that $R(\mathcal{T}_{w'}) \neq [k]$ by choice of v. If r_x is not an endpoint, then there are vertices $r_a, w_1, r_x, w, r_b, r_d$ appearing in order on the cycle for some $r_a, r_b \in R(\mathcal{T}_{w'})$ and $r_d \notin R(\mathcal{T}_{w'})$. In particular, r_x can be connected to r_d via w using vertices outside of $V(\mathcal{T}_{w'})$ and we can connect r_a and r_b using internal vertices in $V(\mathcal{T}_{w'})$. We then apply Lemma 3 to obtain an $\{r_a, r_x, r_b, r_d\}$-rooted K_4-minor, a contradiction.

We now turn to the proof that $\mathcal{T}_w \in \mathcal{B}$. We already proved its roots form an interval. Suppose that w has a single child w'. By the induction hypothesis, $\mathcal{T}_{w'}$ is a partial interval solution. If w is not incident with any roots, then $\mathcal{T}_w$ can be obtained from $\mathcal{T}_{w'}$ with **R2**.

If w is incident with at least one root, we show we can obtain $\mathcal{T}_w$ by gluing the partial interval solution $\mathcal{S}_w = (\{w\}, \emptyset)$ with $\mathcal{T}_{w'}$. To apply **R3**, we need to show all points in the intersection $R(\mathcal{S}_w) \cap R(\mathcal{T}_{w'})$ are endpoints of the corresponding intervals.

- If w is a terminal r_a, then it is not incident with any virtual edges and hence there are no intersection points.
- Suppose that w is not a terminal, but it is incident with either one or two virtual edges $r_{x_1} = ww_2$ and $r_{x_2} = ww_2$. This means $R(\mathcal{S}_w) = \{r_{x_1}, r_{x_2}\}$ and so these are the only possible intersection points. We proved above that if r_{x_j} exists and is in $R(\mathcal{T}_{w'})$, it must be an endpoint of that interval (for $j = 1, 2$). Moreover, each root in $R(\mathcal{S}_w)$ is automatically an endpoint of the interval since the size is 1 or 2.

So we may assume w has at least two children. Let $w_1, \ldots, w_d$ denote the children of w in $\mathcal{T}$ and for each $i \in [d]$, let $\mathcal{T}_i = \mathcal{T}_{w_i}$ denote the subtree of $\mathcal{T}$ containing all descendants of w_i. We already showed that for each $i \in [d]$, $R(\mathcal{T}_i)$ is an interval $R[a_i, b_i]$ for some $a_i, b_i \in [k]$.

The union of these intervals will be $R(\mathcal{T}_w) \setminus \{w\}$. We now describe when $R[a_i, b_i]$ and $R[a_j, b_j]$ can intersect for some $i \neq j$. Every intersection point comes from a virtual edge uu' with $u \in \mathcal{T}_i$ and $u' \in \mathcal{T}_j$, and in particular each root appears in at most two of the $\mathcal{T}_i$. We show that when $r_x \in R[a_i, b_i] \cap R[a_j, b_j]$, we must have $x \in \{a_i, b_i\}$. Indeed, if not, we select

$r_c \notin R[a_i, b_i]$ and now $r_c, r_{a_i}, r_x, r_{b_i}$ appear consecutively on the cycle, with a path between r_{a_i} and r_{b_i} contained within $V(\mathcal{T}_i)$ and a path between r_x and r_c outside of $V(\mathcal{T}_i)$ (via u'), yielding a contradiction via an $\{r_c, r_{a_i}, r_x, r_{b_i}\}$-rooted K_4-minor by Lemma 3.

Moreover, we already argued that $R(\mathcal{S}_w) \cap R[a_i, b_i]$ can only intersect in a_i or b_i (and in fact, at most one of those two by minimality arguments). Combined, this means that there is a way to renumber such that

$$\bigcup_{i=1}^{j} R[a_i, b_i] \cup R(\mathcal{S}_w) \text{ and } \bigcup_{i=j+1}^{d} R[a_i, b_i] \cup R(\mathcal{S}_w)$$

form intervals intersecting only in their endpoints. Here j is chosen so that either $w = r_j$ if w is terminal, and w is between r_{b_j} and $r_{a_{j+1}}$ on the cycle if one (or both) of those is a virtual edge incident to w, and otherwise $R(\mathcal{S}_w) = \emptyset$ and does not affect whether the sets are intervals. ◁

Our dynamic program will ensure that each partial interval solution in $\mathcal{B}$ is considered by following rules **R1**-**R3**. Since these rules are also easily seen to result in a valid Steiner tree, the final output therefore will be optimal by Claim 7.

Base case. All entries are set to ∞ initially. We compute all entries $\mathsf{DP}[a, b, v, s_a, s_b]$ corresponding to a partial interval solution consisting of a single vertex via Observation 5. That is, for each vertex v,

- if v is a terminal r_a, we set $\mathsf{DP}[a, a, v, s_a, s'_a] = 0$;
- if v has exactly one incident virtual edge r_a, we set $\mathsf{DP}[a, a, v, v, v] = w_{r_a}(v)$.
- if v has incident virtual edges r_a and r_{a+1}, we set $\mathsf{DP}[a, a+1, v, v, v] = w_{r_a}(v) + w_{r_{a+1}}(v)$.

Adding a vertex. We first implement **R2**: adding a vertex not incident to any roots. Suppose that v is not incident with any roots. For any edge $uv \in E$, we add the rule

$$\mathsf{DP}[a, b, v, s_a, s_b] \leftarrow \min(\mathsf{DP}[a, b, u, s_a, s_b] + w_{uv}, \mathsf{DP}[a, b, v, s_a, s_b]).$$

Next, we implement **R3**, "gluing on v", which has multiple cases because we also need to properly keep track of the costs of the virtual edges.

Gluing two solutions without intersection. We first consider the case in which the intervals of the roots are disjoint, in which case the partial interval solutions that we are gluing do not have any virtual edges between them. When $a_2 = b_1 + 1$ and $a_1 \notin [a_2, b_2]$, we set

$$\mathsf{DP}[a_1, b_2, v, s_{a_1}, s_{b_2}] \leftarrow \min(\mathsf{DP}[a_1, b_2, v, s_{a_1}, s_{b_2}], \mathsf{DP}[a_1, b_2, v, s_{a_1}, s_{b_2}] + \mathsf{DP}[a_2, b_2, v, s_{a_2}, s_{b_2}]).$$

Gluing two solutions with one intersection. Now suppose that the intervals overlap in exactly one point.

When $a_2 = b_1$ with r_{a_2} a terminal and $a_1 \notin [a_2 + 1, b_2]$, we set

$$\mathsf{DP}[a_1, b_2, v, s_{a_1}, s_{b_2}] \leftarrow \min(\mathsf{DP}[a_1, b_2, v, s_{a_1}, s_{b_2}], \mathsf{DP}[a_1, b_1, v, s_{a_1}, s_{b_1}] + \mathsf{DP}[a_2, b_2, v, s_{a_2}, s_{b_2}]).$$

We may forget about s_{a_2}, s_{b_1} since the "status" of the terminal is irrelevant to us.

When $r_{a_2} = uu'$ is a virtual edge, we need to consider various options, depending on the status this edge used to be in for both solutions, and what final state we want it to be, and whether $a_1 = b_1$ and/or $a_2 = b_2$.

We start with when we want to make the status d. We need to ensure u, u' are part of the solution, and the solutions must overlap elsewhere in some vertex v. So when $a_2 = b_1$, $a_1 \notin [a_2 + 1, b_2]$, we set

$$\begin{aligned}\mathsf{DP}[a_1, b_2, v, x_1, x_2] \leftarrow \min(&\mathsf{DP}[a_1, b_2, v, x_1, x_2],\\ &\mathsf{DP}[a_1, b_1, v, s_{a_1}, u] + \mathsf{DP}[a_2, b_2, v, u', s_{b_2}] - w_{uu'}(u) - w_{uu'}(u') + w_{uu'}(\mathsf{d})),\\ &\mathsf{DP}[a_1, b_1, v, s_{a_1}, u'] + \mathsf{DP}[a_2, b_2, v, u, s_{b_2}] - w_{uu'}(u) - w_{uu'}(u') + w_{uu'}(\mathsf{d})),\end{aligned}$$

where we require $x_1 = s_{a_1}$ if $a_1 \neq b_1$ and $x = \mathsf{d}$ when $a_1 = b_1$; and we require $x_2 = s_{b_2}$ if $a_2 \neq b_2$ and $x_2 = \mathsf{d}$ when $a_2 = b_2$. In the last line, for example, we use that the first partial interval solution has paid the cost $w_{uu'}(u')$ (as recorded by the status) and the second $w_{uu'}(u)$. We allow here to replace the cost by $w_{uu'}(\mathsf{d})$, since the solutions can be merged via the vertex v. If $a_1 = b_1$ or $a_2 = b_2$, we will keep the status d recorded at the relevant endpoint.

We also give the option to "connect" via the virtual edge r_{a_2}. When $r_{a_2} = uu'$ is a virtual edge, $a_2 = b_1$ $a_1 \notin [a_2 + 1, b_2]$, we set

$$\begin{aligned}\mathsf{DP}[a_1, b_2, u, x_1, x_2] \leftarrow \min(&\mathsf{DP}[a_1, b_2, u, x_1, x_2],\\ &\mathsf{DP}[a_1, b_1, u, s_{a_1}, u] + \mathsf{DP}[a_2, b_2, u', u', s_{b_2}] - w_{uu'}(u) - w_{uu'}(u') + w_{uu'}(\mathsf{c})),\\ \mathsf{DP}[a_1, b_2, u', x_1, x_2] \leftarrow \min(&\mathsf{DP}[a_1, b_2, u', x_1, x_2],\\ &\mathsf{DP}[a_1, b_1, u, s_{a_1}, u] + \mathsf{DP}[a_2, b_2, u', u', s_{b_2}] - w_{uu'}(u) - w_{uu'}(u') + w_{uu'}(\mathsf{c})),\end{aligned}$$

where again, we require $x_i = s_{a_i}$ if $a_i \neq b_i$ and $x_i = \mathsf{c}$ otherwise.

To allow the edge to be in status u (or u', analogously), we also add

$$\begin{aligned}\mathsf{DP}[a_1, b_2, v, x_1, x_2] \leftarrow \min(&\mathsf{DP}[a_1, b_2, v, x_1, x_2],\\ &\mathsf{DP}[a_1, b_1, v, s_{a_1}, u'] + \mathsf{DP}[a_2, b_2, v, u, s_{b_2}] - w_{uu'}(u'),\\ &\mathsf{DP}[a_1, b_1, v, s_{a_1}, u] + \mathsf{DP}[a_2, b_2, v, u, s_{b_2}] - w_{uu'}(u),\\ &\mathsf{DP}[a_1, b_1, v, s_{a_1}, u] + \mathsf{DP}[a_2, b_2, v, u', s_{b_2}] - w_{uu'}(u')),\end{aligned}$$

where again, we require $x_i = s_{a_i}$ if $a_i \neq b_i$ and $x_i = \mathsf{c}$ otherwise.

Gluing two solutions with two intersections It remains now to glue intervals with two intersection points to a final solution. The rules for this are analogous to the previous case, but now the compatibility is checked at both endpoints, $a_1 = b_2$ and $a_2 = b_1$.

We apply the rules in a bottom-up fashion. The final output is the minimum over all entries $\mathsf{DP}[a, b, v, s_a, s_b]$ with $R[a, b] = R$.

Running time. There are $O(n^3)$ entries, each of which takes $O(n)$ time to compute. Thus, the total running time is $O(n^4)$. ◀

4.3 Preprocessing

Throughout, the algorithm works on a simple graph G^* (that is, at most one edge per pair of vertices). We can always obtain this via the following *edge pruning* steps.

1. If there are two edges e and e' containing the same two endpoints with $w_e \leq w_{e'}$, then delete e'.

2. If there are two virtual edges e_1^* and e_2^* over the same endpoints u and v, then delete e_1^* and e_2^* add a new virtual edge e^* between u and v with the virtual weight defined as follows:
   - $w_{e^*}(u) = w_{e_1^*}(u) + w_{e_2^*}(u)$.
   - $w_{e^*}(v) = w_{e_1^*}(v) + w_{e_2^*}(v)$.
   - $w_{e^*}(\mathsf{c}) = \min(w_{e_1^*}(\mathsf{c}) + w_{e_2^*}(\mathsf{d}), w_{e_1^*}(\mathsf{d}) + w_{e_2^*}(\mathsf{c}))$.
   - $w_{e^*}(\mathsf{d}) = w_{e_1^*}(\mathsf{d}) + w_{e_2^*}(\mathsf{d})$.
3. If there is an edge e and virtual edge e^* over the same endpoints say u and v, then delete e, and update the virtual weight of e^* by $w_{e^*}(\mathsf{c}) = \min(w_{e^*}(\mathsf{c}), w_{e^*}(\mathsf{d}) + w_e)$.

We briefly discuss the correctness of the preprocessing steps above, though all of them follow directly from the definition of Virtual Edge Steiner Tree. Firstly, note that we exactly remove all multiple edges (and preserve which vertices have at least one edge between them). For (1), an edge with a larger weight is never included in an optimal solution so can be removed. For (2), at most one of the two virtual edges will be used to "connect" the vertices, but we still need to pay the cost for both even if they are not "included" in the solution. Hence the new edge has the weight defined as the sum of the weight of two virtual edges for the cases u, v, and d. For the case c, we may assume that exactly one of e_1^* and e_2^* is included into the solution. Lastly, for (3), we update w_{e^*} to take into account that it may be cheaper to connect via the edge e in the scenario that u and v needs to be connected. That is, if there is an edge e and virtual edge e^* such that $w_{e^*}(\mathsf{c}) \geq w_{e^*}(\mathsf{d}) + w_e$, then we update $w_{e^*}(\mathsf{c})$ to $w_{e^*}(\mathsf{d}) + w_e$. In the other scenarios an optimal solution would never add the edge e.

We define a preprocessing procedure as follows to ensure that every biconnected component has at least one root and that every triconnected component has at least two roots.

We say a vertex set $A \subseteq V$ is *incident with a root* if $A \cap T \neq \emptyset$ or there is a virtual edge incident with a vertex of A.

1. Perform edge pruning.
2. If there is a cut vertex v such that $G - v$ has a connected component A that is not incident with any roots, then delete A from the graph. Return to 1.
3. If there is a 2-cut $\{u, v\}$ such that $G[V \setminus \{u, v\}]$ has a connected component A that is not incident with a root, then delete A from the graph and add an edge uv, where the weight w_{uv} equals $\mathrm{dist}_{G[A \cup \{u,v\}]}(u, v)$. Return to 1.
4. If there is a 2-cut $\{u, v\}$ such that $G[V \setminus \{u, v\}]$ has a connected component A that is incident with exactly one root, we compute the weights for a new virtual edge $\{u, v\}$ by solving four Virtual Edge Steiner Tree instances using Observation 5.
   - $w_{uv}(u)$ is the cost of the instance induced by $A \cup \{u\}$ with u as an additional terminal.
   - $w_{uv}(v)$ is the cost of the instance induced by $A \cup \{v\}$ with v as an additional terminal.
   - $w_{uv}(\mathsf{c})$ is the cost of the instance induced by $A \cup \{u, v\}$ with $\{u, v\}$ as terminals but where we do not include a virtual edge between u and v if it had one.
   - $w_{uv}(\mathsf{d}) = \min(w_{uv}(u), w_{uv}(v))$.

   We add a new virtual edge $\{u, v\}$ with the costs as defined above and remove all vertices from A. (If there was already a virtual edge between u and v, we keep it and it will be dealt with during the cleaning.) Return to 1.

For (2), note that since v is a cut vertex, there is always an optimal solution that has empty intersection with A.

For (3), if u and v are connected in an optimal solution via A, then this will be done via a shortest path between u and v.

Algorithm 1 A polynomial-time algorithm for Virtual Edge Steiner Tree. We assume that the input graph is connected and that there is no rooted K_4-minor in the instance.

procedure Algo$((G, T, E^*, w_e))$
  Apply preprocessing from Section 4.3
  if $|T| + |E^*| = O(1)$ **then return** Solve by Observation 5 and Dreyfus-Wagner.
  if G^* is not 2-connected **then**
    Let v be a cut vertex that separates G into A and B.
    return Algo$(G[A \cup \{v\}], (T \cap A) \cup \{v\}, E^* \cap \binom{A}{2}, w)$
      + Algo$(G[B \cup \{v\}], (T \cap B) \cup \{v\}, E^* \cap \binom{B}{2}, w)$.
  else if G^* is not 3-connected **then**
    Let $\{u, v\}$ be a cut that separates G into A and B with $|A| \leq |B|$.
    Let e be a virtual edge connecting u and v.
    $w_e(u) \leftarrow$ Algo$(G[A \cup \{u\}], (T \cap A) \cup \{u\}, E^* \cap \binom{A}{2}, w)$
    $w_e(v) \leftarrow$ Algo$(G[A \cup \{v\}], (T \cap A) \cup \{v\}, E^* \cap \binom{A}{2}, w)$
    $w_e(\mathsf{c}) \leftarrow$ Algo$(G[A \cup \{u, v\}], (T \cap A) \cup \{u, v\}, E^* \cap \binom{A}{2}, w)$
    $w_e(\mathsf{d}) \leftarrow$ Algo$(G[A \cup \{u, v\}] + uv, (T \cap A) \cup \{u, v\}, E^* \cap \binom{A}{2}, w)]$
    return Algo$(G[B \cup \{u, v\}], T, E^* \cup \{e\}, w)$
  else
    return the result for 3-connected case.

For (4), there are four cases to consider. In the first three cases, we enforce u, v or both u and v are contained in the optimal Steiner tree by making them terminals. In the last, we note that the solution on A is allowed to go either via u or via v. Note that the instances that we solve to define the costs all have at most 3 roots (virtual edges or terminals) and so can be solved in polynomial time in terms of $|A|$ using Observation 5.

4.4 Our algorithm

See Algorithm 1 for the outline of the algorithm. Our algorithm maintains the invariant that there is no rooted K_4-minor over the terminals T and virtual edges E^*. We first apply the preprocessing steps in Section 4.3. If this results in a graph with $O(1)$ roots, we solve the problem by reducing to Steiner Tree with $O(1)$ terminals, which can then be solved using the Dreyfus-Wagner algorithm [8] in $O(1)$ time.

***G** is not 2-connected.** Let v be a cut vertex separating G into A and B. (Here, A and B each may contain multiple connected components of $G - v$.) We recursively solve two instances induced on vertex sets $A \cup \{v\}$ and $B \cup \{v\}$ where v is an additional terminal. To see why this recursion is correct, note that A and B each contains at least one terminal or virtual edge by the preprocessing steps. So, any Steiner tree must cover v as well, and thus it is obtained by "gluing" the two solutions on the vertex v. The total cost equals the sum of the two recursive instance costs.

We claim that no rooted K_4-minor is introduced in the recursive calls. As v is a cut vertex, there is a path from v to each vertex in B. Moreover, by the preprocessing, B contains a root (there is either a virtual edge in $B \cup \{v\}$ or a terminal in B). This means that any rooted K_4-minor in the instance on $A \cup \{v\}$ with v as terminal also gives rise to a rooted K_4-minor in G.

G^* is not 3-connected. We first find a 2-cut $\{u, v\}$ that separates the graph into A and B with $|A| \leq |B|$. Our algorithm considers four cases based on whether u and v are covered by the solution.

- We solve the subproblem over $G[A \cup \{u, v\}]$ with u as an additional terminal, to account for the case in which u is covered but v is not.
- We solve the subproblem over $G[A \cup \{u, v\}]$ with v as an additional terminal, to account for the case in which v is covered but u is not.
- We solve the subproblem over $G[A \cup \{u, v\}]$ with u and v as additional terminals. This accounts for the case that both u and v are covered, and they are connected through A.
- We solve the subproblem over $G[A \cup \{u, v\}]$ with u as terminal and the edge $\{u, v\}$ added (in E) with cost $w(\{u, v\}) = 0$. This accounts for the case in which both u and v are covered, and they are not connected through A.

Next, we solve a single instance on vertex set $B \cup \{u, v\}$ where $\{u, v\}$ is added as virtual edge (in E^*). The cost of the virtual edge $\{u, v\}$ is determined by the previous calculations

We verify that no rooted K_4-minor is introduced in the recursive calls. The first two recursive calls (corresponding to the cases u and v) are analogous to the scenario where G^* has a cut vertex. For the third and fourth calls (cases c and d), assume for contradiction that there exists a rooted K_4-minor over $T \cup \{u, v\}$ in $G[A \cup \{u, v\}] + uv$. Our preprocessing steps ensure two roots $r, r' \in T \cup E^*$. Since there is no cut vertex, there are two disjoint paths connecting $\{r, r'\}$ to $\{u, v\}$. Using these paths, we can construct a rooted K_4-minor in G, which is a contradiction. For the final recursive call on $B \cup \{u, v\}$, the argument for the case where G^* has a cut vertex again shows the non-existence of a rooted K_4-minor.

G^* is 3-connected. If G^* is 3-connected, then we can find a polynomial time a cycle going through all roots by Lemma 4 presented in Section 3, and Virtual Edge Steiner Tree can be solved as described in Section 4.2

Runtime analysis

When we solve an instance with G^* being 3-connected, we already showed the running time is at most $c'n^4$ for some constant $c' > 0$, assuming $n \geq n_0$ for some constant n_0. After making c' larger if needed, we can also assume all additional steps in the algorithm such as finding a 2-cut and preprocessing take at most $c'n^3$ on inputs of $n \geq n_0$ vertices.

We let $T(n)$ denote the maximal running time on an input graph on n vertices and show that $T(n) \leq cn^4$ for some constant $c > 0$ by induction on n. We will choose c such that $c \geq c'$ and $T(n) \leq cn^4$ when $n \leq n_0$.

When G^* is not 2-connected, we find a cut vertex v splitting the graph into A and B and recursively solve two instances on $A \cup \{v\}$ and $B \cup \{v\}$ and obtain the cost from there. So with $|A| = i$, we find the running time is at most $T(i + 1) + T(n - i) + c'n^3$.

When G^* is 2-connected, but not 3-connected, we find a 2-cut $\{u, v\}$ that separates G^* into A and B with $|A| \leq |B|$, and therefore $|A| \leq (n - 2)/2$. With $i = |A|$, we recursively solve four instances of size at most i and one instance of size $|B| \leq n - i$, resulting in a running time of at most

$$\max_{1 \leq i \leq n/2} T(n - i) + 4T(i + 2) + c'n^3.$$

Applying the inductive hypothesis, we find $T(n - i) \leq c(n - i)^4$ and $T(i + 2) \leq c(i + 2)^4$ for all $1 \leq i \leq n/2$. The function $f(i) = c(n - i)^4 + 4c(i + 2)^4 + c'n^3$ is convex within the domain

$1 \leq i \leq n/2$ because the second derivative $f''(i)$ (with respect to i) is positive. Therefore, the maximum value is attained either at $i = 1$ or $i = n/2$. Evaluating these points,

$$f(1) = c(n-1)^4 + 4c(3^4) + c'n^3$$
$$f(n/2) = c(n/2)^4 + 4c(n/2+2)^4 + c'n^3,$$

we see both are less than cn^4 when n is sufficiently large since $c \geq c'$. Hence, we conclude that $T(n) = O(n^4)$.

References

1 Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994. doi:10.1145/174644.174650.

2 Marshall W. Bern and Daniel Bienstock. Polynomially solvable special cases of the Steiner problem in planar networks. *Ann. Oper. Res.*, 33(6):403–418, 1991. doi:10.1007/BF02071979.

3 Thomas Böhme, Jochen Harant, Matthias Kriesell, Samuel Mohr, and Jens M. Schmidt. Rooted minors and locally spanning subgraphs. *J. Graph Theory*, 105(2):209–229, 2024. doi:10.1002/JGT.23012.

4 Glencora Borradaile, Philip N. Klein, and Claire Mathieu. An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. *ACM Trans. Algorithms*, 5(3):31:1–31:31, 2009. doi:10.1145/1541885.1541892.

5 Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, and Maximilian Probst Gutenberg. Almost-linear time algorithms for incremental graphs: Cycle detection, sccs, s-t shortest path, and minimum-cost flow. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC*, pages 1165–1173. ACM, 2024. doi:10.1145/3618260.3649745.

6 Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005. doi:10.1145/1101821.1101823.

7 R. Diestel. *Graph Theory: 5th edition*. Springer Graduate Texts in Mathematics. Springer Berlin, Heidelberg, 2017. URL: https://books.google.nl/books?id=zIxRDwAAQBAJ.

8 Stuart E. Dreyfus and Robert A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971. doi:10.1002/NET.3230010302.

9 Ranel E. Erickson, Clyde L. Monma, and Arthur F. Veinott Jr. Send-and-split method for minimum-concave-cost network flows. *Math. Oper. Res.*, 12(4):634–664, 1987. doi:10.1287/MOOR.12.4.634.

10 Ruy Fabila-Monroy and David R. Wood. Rooted K_4-Minors. *Electronic Journal of Combinatorics*, 20(2):#P64, 2013. doi:10.37236/3476.

11 Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956.

12 Jędrzej Hodor, Hoang La, Piotr Micek, and Clément Rambaud. Quickly excluding an apex-forest, 2024.

13 Ken ichi Kawarabayashi. Rooted minor problems in highly connected graphs. *Discrete Mathematics*, 287(1):121–123, 2004. doi:10.1016/j.disc.2004.07.007.

14 Leif K Jørgensen and Ken-ichi Kawarabayashi. Extremal results for rooted minor problems. *Journal of Graph Theory*, 55(3):191–207, 2007. doi:10.1002/jgt.20232.

15 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce A. Reed. The disjoint paths problem in quadratic time. *J. Comb. Theory B*, 102(2):424–435, 2012. doi:10.1016/J.JCTB.2011.07.004.

16 Tuukka Korhonen, Michał Pilipczuk, and Giannos Stamoulis. Minor containment and disjoint paths in almost-linear time. *arXiv preprint arXiv:2404.03958*, 2024.

17 Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. On subexponential parameterized algorithms for Steiner tree and directed subset TSP on planar graphs. In *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, pages 474–484. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00052.

18 Kazuhiko Matsumoto, Takao Nishizeki, and Nobuji Saito. An efficient algorithm for finding multicommodity flows in planar networks. *SIAM J. Comput.*, 14(2):289–302, 1985. doi:10.1137/0214023.

19 Marcin Pilipczuk, Michal Pilipczuk, Piotr Sankowski, and Erik Jan van Leeuwen. Network sparsification for Steiner problems on planar and bounded-genus graphs. *ACM Trans. Algorithms*, 14(4):53:1–53:73, 2018. doi:10.1145/3239560.

20 J. Scott Provan. An approximation scheme for finding Steiner trees with obstacles. *SIAM J. Comput.*, 17(5):920–934, 1988. doi:10.1137/0217057.

21 Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

22 Paul Wollan. Extremal functions for rooted minors. *Journal of Graph Theory*, 58(2):159–178, 2008. doi:10.1002/jgt.20301.

23 David R. Wood and Svante Linusson. Thomassen's choosability argument revisited. *SIAM Journal on Discrete Mathematics*, 24(4):1632–1637, 2010. doi:10.1137/100796649.