Discrete Applied Mathematics 157 (2009) 673–684

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Shortest paths in linear time on minor-closed graph classes, with an application to Steiner tree approximation

Siamak Tazari [a,*], Matthias Müller-Hannemann [b]

[a] Technische Universität Darmstadt, Department of Computer Science, Hochschulstraße 10, 64289 Darmstadt, Germany

[b] Martin-Luther-Universität Halle-Wittenberg, Department of Computer Science, Von-Seckendorff-Platz 1, 06120 Halle (Saale), Germany

ARTICLE INFO

Article history:
Received 31 October 2006
Received in revised form 30 April 2008
Accepted 7 August 2008
Available online 7 September 2008

Keywords:
Shortest path
Minor-closed graph classes
Planar graph
Steiner tree
Mehlhorn's algorithm

ABSTRACT

We generalize the linear-time shortest-paths algorithm for planar graphs with nonnegative edge-weights of Henzinger et al. (1994) to work for any proper minor-closed class of graphs. We argue that their algorithm can not be adapted by standard methods to all proper minor-closed classes. By using recent deep results in graph minor theory, we show how to construct an appropriate recursive division in linear time for any graph excluding a fixed minor and how to transform the graph and its division afterwards, so that it has maximum degree three. Based on such a division, the original framework of Henzinger et al. can be applied. Afterwards, we show that using this algorithm, one can implement Mehlhorn's (1988) 2-approximation algorithm for the Steiner tree problem in linear time on these graph classes.

1. Introduction

The single-source shortest-paths problem with nonnegative edge-weights is one of the most-studied problems in computer science, because of both its theoretical and practical importance. Dijkstra's classical algorithm [1] has ever since its discovery been one of the best choices in practice. Also from a theoretical point of view, until very recently, it had the best running time in the addition-comparison model of computation, namely $O(m + n \log n)$ using Fibonacci heaps [2] (we use n to denote the number of vertices of a graph and m for its number of edges). Pettie and Ramachandran [3] improved the theoretical running time in undirected graphs for the case when the ratio r between the largest and smallest edge-weight is not too large. They achieve a running time of $O(m\alpha(m, n) + \min\{n \log n, n \log\log r\})$, where $\alpha(m, n)$ is the very slowly growing inverse-Ackermann function. Goldberg [4] proposed an algorithm that runs on average in linear time. For the case of integer edge-weights, Thorup [5] presented a linear-time algorithm in the word RAM model of computation, where the bit-manipulation of words in the processor is allowed. Hagerup [6] extended and simplified Thorup's ideas to work for directed graphs in nearly linear time. But the question whether the standard addition-comparison model allows shortest-paths computation in worst-case linear-time is still open. For a fairly recent survey about shortest-paths algorithms, see Zwick [7].

For planar graphs, Henzinger et al. [8] presented the first linear-time algorithm to calculate shortest-paths with nonnegative edge-weights. Their algorithm works on directed graphs. It is based on Frederickson's [9,10] work who gave an $O(n\sqrt{\log n})$-time algorithm for this case and whose idea was in turn based on Lipton and Tarjan's planar separator [11] to decompose the graph. Henzinger et al. first decompose the graph into a recursive division and then use this division to relax the edges in a certain order that guarantees linear running time. They claim that their algorithm can be adapted to

* Corresponding author. Tel.: +49 6151 163358; fax: +49 6151 164025.
E-mail addresses: tazari@algo.informatik.tu-darmstadt.de (S. Tazari), muellerh@informatik.uni-halle.de (M. Müller-Hannemann).

0166-218X/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2008.08.002

work for any proper minor-closed family of graphs where small separators can be found in linear time. Recently, Reed and Wood [12] improved the quadratic-time separator of Alon et al. [13] and showed that all proper minor-closed graph classes can be separated in linear time; so, we should be done. *However, both Frederickson's algorithm and Henzinger et al.'s algorithm assume that the graph has maximum degree* 3; *while this property can be achieved easily for planar graphs, we argue that it can not be achieved by standard methods for arbitrary minor-closed classes* (in particular, it can not be applied to apex graphs, i.e. planar graphs augmented by a "super-source"; these graphs have frequent application in the literature). We show how to build an appropriate recursive division of a graph from a proper minor-closed family in linear time by a nontrivial extension of the algorithm in [8]. Our algorithm works for graphs with arbitrary degrees. But even after having the recursive division, the shortest paths algorithm in [8] depends on the assumption that the graph has bounded degree (and contains only a single source labeled initially with distance zero, cf. apex graphs). Using our recursive division, we show how to transform the graph and its division to have maximum degree 3, so that Henzinger et al.'s shortest-paths algorithm can be applied. Our modifications lead to the first shortest-paths algorithm for all proper minor-closed classes of graphs that runs in linear time in the addition-comparison model of computation.

We also consider the Steiner tree problem, namely finding the shortest tree that connects a given set of terminals in an undirected graph. The Steiner tree problem is also one of the most fundamental problems in computer science and of the first problems shown to be $\mathcal{NP}$-complete by Karp [14]. Bern and Plassmann [15] showed that it is even $\mathcal{APX}$-hard and the best-known nonapproximability result is due to Chlebík and Chlebíková [16] who showed a bound of $96/95 \approx 1.01053$. Robins and Zelikovsky [17] presented an algorithm with approximation guarantee $1 + \frac{\ln 3}{2} + \epsilon \approx 1.55 + \epsilon$ which is the best approximation algorithm for this problem known so far. There is a well-known 2-approximation algorithm for this problem [18,19] that is based on finding the minimum spanning tree of the complete distance network of the set of terminals. Mehlhorn [20] improved the running time of this algorithm to $O(m + n \log n)$.

The Steiner tree problem in planar graphs is also $\mathcal{NP}$-hard [21] but very recently a polynomial time approximation scheme (PTAS) has been found by Borradaile et al. [22,23] for this case. The running time of the PTAS is $O(n \log n)$ with a constant factor that is exponential in the inverse of the desired accuracy. As an application of our shortest-paths algorithm, we show how to implement Mehlhorn's [20] 2-approximation algorithm in linear time on proper minor-closed graph classes. No better time bound than Mehlhorn's own implementation of $O(m + n \log n)$ has previously been known even for planar graphs. An important observation that we made is that Mehlhorn's distance network is a minor of the given graph and thus, its minimum spanning tree can be calculated in linear time with the algorithm of Mares [24] (or that of Cheriton and Tarjan [25] in the planar case).

Our contribution and outline

The area of graph minor theory has been constantly evolving ever since the graph minor theorem of Robertson and Seymour [26] was announced in 1988. Many important algorithms and meta-algorithms have been presented for large problem families on minor-closed graph classes and numerous theoretical concepts have been developed to handle them. We present the first linear-time algorithms for two fundamental graph-theoretic problems in these classes.

Our contribution can be summarized as follows:

- *identifying* that there is a gap in generalizing Henzinger et al.'s recursive division and shortest paths algorithms to all proper minor-closed graph classes;
- arguing that the gap *can not* be closed by standard methods;
- *filling in* the gap using deep results in graph minor theory;
- (re)proving in detail the *correctness* of the modified algorithm;
- showing how to modify a graph and its recursive division to obtain a bounded-degree graph and a recursive division with the same properties, resulting in the *first linear-time shortest-paths algorithm* for proper minor-closed graph classes;
- obtaining a useful application, namely, the *first linear time Steiner tree approximation*, which was previously not even known for planar graphs.

In Section 2, we review some needed concepts and previous work; in Section 3, we present our main result about shortest paths and in Section 4, the application to Steiner tree approximation.

2. Preliminaries

In this section, we review some concepts and some previous results that are needed in this work. These include graph minors, vertex partitioning, graph decomposition, and Henzinger et al.'s [8] single-source shortest-paths algorithm.

2.1. Graph minors

A *minor* of a graph G is a graph that is obtained from a subgraph of G by contracting a number of edges. A class of graphs is *minor-closed* if it is closed under building minors. It is called a *proper* class if it is neither empty nor the class of all graphs. Examples of proper minor-closed graph families are planar graphs, bounded-genus graphs, and apex graphs. The seminal theorem of Robertson and Seymour [26] states that any proper minor-closed class of graphs can be characterized by a finite

set of excluded minors. Note that for a proper minor-closed class of graphs, we can always consider the number of vertices ℓ of the smallest excluded minor and conclude that the complete graph K_ℓ is a particular excluded minor of the class. Thus, the class of K_ℓ-minor-free graphs includes the considered minor-closed class of graphs. In the rest of this work, we work with K_ℓ-minor-free graphs, where ℓ is a fixed constant.

It follows from a theorem of Mader [27] that K_ℓ-minor-free graphs have constant average degree, for some constant depending on ℓ. This, in turn, implies that these classes of graphs are sparse, i.e. we have $m = O(n)$.

2.2. Vertex partitioning

In [9], Frederickson presented a simple algorithm called `FindClusters`, based on depth-first search, that given a parameter z and an undirected graph *with maximum degree* 3, partitions its vertices into *connected components* each having at least z and at most $3z$ vertices. Note that since the algorithm gives us connected components, we can contract each one of them and get a minor of the input graph with at most n/z vertices. Frederickson used this algorithm to derive fast algorithms for the minimum spanning tree and shortest-paths [10] problems. If a weighted graph does not have maximum degree 3, one can apply the following transformation: replace a vertex v of degree $d(v)$ with a zero-weight path of length $d(v)$, such that each edge incident to v is now incident to exactly one vertex of the path, i.e. we can *split* v using zero-weight edges. A similar transformation can be applied to directed graphs, too, using an additional zero-weight edge to complete a directed cycle. If the given graph is embedded in a surface, one can order the edges around the path/cycle in the same way they were ordered around the corresponding vertex in the given embedding. This way, the transformed graph will also be embedded in the same surface. However, for an arbitrary minor-closed class of graphs (e.g. apex graphs), it might not always be possible to remain in the class after transforming the graph this way, see Section 3. But Frederickson's `FindClusters` *depends* on the graph having bounded degree. Any constant bound would suffice for our purposes but in general such a bound does not exist for arbitrary minor-closed graph families.

Reed and Wood [12] introduced an alternative partitioning concept that can be applied to a graph $G = (V, E)$ with arbitrary degrees excluding a fixed minor. Consider some partitioning $\mathcal{P} = \{P_1, \ldots, P_t\}$ of the vertex set V. Let $H = (V_H, E_H)$ be the graph obtained by collapsing every part P_i of G into a single vertex $v_i \in V_H$ $(1 \leq i \leq t)$ and removing loops and parallel edges. This way, there is an edge between two vertices v_i and v_j of H if and only if there is an edge between a vertex of P_i and a vertex of P_j in G $(1 \leq i < j \leq t)$. We say $\mathcal{P}$ is a *connected H-partition* of G if $v_iv_j \in E_H$ if and only if there is an edge of G between every connected component of P_i and every connected component of P_j. Reed and Wood proved the following lemma[1]:

Lemma 2.1 ([12]). *There is a linear-time algorithm that given a constant z and a graph G excluding a fixed K_ℓ-minor, outputs a connected H-partition $\mathcal{P} = \{P_1, \ldots, P_t\}$ of G such that $t \leq n/z$, and $|P_i| < c_0 \cdot z$ for all $1 \leq i \leq t$, where c_0 is a constant depending only on ℓ.*

Note that by contracting each connected component of each P_i in G to a single vertex, one gets a graph that contains an isomorphic copy of H as a subgraph and so, H is a minor of G and in particular, is also K_ℓ-minor-free. Hence, when dealing with graphs with no bounded degree, Lemma 2.1 can be used instead of `FindClusters` to partition the graph and reduce its size while keeping it free of some fixed minor.

Corollary 2.2. *Let G be a graph with n vertices excluding a fixed K_ℓ-minor, and let $c_0 = 2^{\ell^2+\ell}$ be a fixed constant. There exists a linear-time algorithm H-`Partition`(G, z, ℓ) with the following properties:*

- *it partitions the vertices of G into at most n/z sets;*
- *each set has at most $c_0 z$ vertices;*
- *it collapses each set into a single vertex, creating a new graph G';*
- *G' is a minor of G with at most n/z vertices.*

2.3. Graph decomposition

A *balanced vertex-separation* of a graph $G = (V, E)$ is given by two sets A and B, such that $A \cup B = V$, there is no edge between $A \setminus B$ and $B \setminus A$, and each one of A and B contains at most an α-fraction of the vertices (for some $1/2 \leq \alpha < 1$). The *size* of the separation is $|A \cap B|$. For a function f, a subgraph-closed class of graphs is said to be *f-separable* if every n-vertex graph in the class has an $O(f(n))$-size separator. Reed and Wood [12] showed that all K_ℓ-minor-free graphs are f-separable in *linear* time for $f(n) = O(n^{2/3})$. For planar graphs, one can use the seminal planar separator theorem of Lipton and Tarjan [11] that delivers an $O(\sqrt{n})$-separator in linear time.

An (r, s)-*division* of an n-vertex graph is a partition of the edges of the graph into $O(n/r)$ regions, each containing $r^{O(1)}$ vertices and each having at most s *boundary vertices* (i.e. vertices that occur in more than one region). For a nondecreasing

[1] In their lemma, we substitute $c_0 := 2^{\ell^2+\ell}$ and $z := 2k/c_0$.

positive integer function f and a positive integer sequence $\bar{r} = (r_0, r_1, \ldots, r_k)$, an $(\bar{r}, f)$-*recursive division* of an n-vertex graph is defined as follows: it contains one region R_G consisting of all of G. If G has more than one edge and $\bar{r}$ is not empty, then the recursive division also contains an $(r_k, f(r_k))$-division of G and an $(\bar{r}', f)$-recursive division of each of its regions, where $\bar{r}' = (r_0, r_1, \ldots, r_{k-1})$. A recursive division can be represented compactly by a *recursive division tree*, a rooted tree whose root represents the whole graph and whose leaves represent the edges of the graph. Every internal node represents a region, namely, the region induced by all the leaves in its subtree. The children of a node of the tree are its immediate subregions in the recursive division.

Using Frederickson's partitioning [9] and division [10] methods, Henzinger et al. [8] present a linear-time algorithm to find certain recursive divisions in planar graphs: they determine a vector $\bar{r}$ and an $(\bar{r}, cf)$-recursive division of the graph for some constant c, such that the inequality

$$\frac{r_i}{f(r_i)} \geq 8^i f(r_{i-1}) \log r_{i+1} \left(\sum_{j=1}^{i+1} \log r_j \right) \tag{1}$$

is satisfied for all r_i's exceeding a constant. The obtained recursive division tree has $O(n)$ nodes and its depth is roughly $O(\log^\star n)$. The idea of the algorithm is as follows: first, iteratively reduce the size of the graph by partitioning the vertices of the graph (using Frederickson's `FindClusters`) and building minors; then, working backwards, find (r, s)-divisions of the smaller graphs (for appropriate values of r and s), imposing divisions on the larger graphs and at the same time building the recursive division tree. Since the time-consuming calculation of (r, s)-divisions is done on the smaller graphs, they succeed to prove that the overall time complexity is linear.

2.4. Single-source shortest-paths on planar graphs

Henzinger et al. prove the following theorem:

Theorem 2.3 (*[8]*). *Let a graph G be given with maximum in-/outdegree 2 and assume that G is equipped with an $(\bar{r}, cf)$-recursive division tree, for some constant c, so that inequality (1) is satisfied for all r_i's exceeding a constant. Then, the single-source shortest-paths problem with nonnegative edge-weights can be solved on G in linear time.*

To prove this theorem, they use a complicated charging scheme that also depends on the graph having *a single source and bounded degree*. Together with the result from the previous subsection, it follows that single-source shortest-paths with nonnegative edge-weights can be calculated in linear-time on planar graphs.

3. Single-source shortest paths on minor-closed graph classes

In this section, we prove our main theorem about shortest paths:

Theorem 3.1. *In every proper minor-closed class of graphs, single-source shortest-paths with nonnegative edge-weights can be calculated in linear time.*

First, we argue that the degree requirement of Henzinger et al.'s algorithm can not be fulfilled by standard methods for arbitrary minor-closed classes of graphs. By "standard methods" we mean splitting a vertex using zero-weight edges until the desired degree bound is reached. In Section 2.2 we discussed a particular way of splitting vertices that can be applied to embedded graphs. In this section, we show that there exist K_ℓ-minor-free graphs, so that *no matter how we split the vertices*, the resulting graph will include a minor whose size can not be bounded by a function in ℓ. The key lies in the observation that splitting an apex might introduce arbitrarily large minors. This is a well-known fact in graph minor theory [28]. For the sake of completeness, we include a short proof below. Apices are a fundamental part of minor-closed graph classes as is demonstrated by the powerful graph-decomposition theorem of Robertson and Seymour [28]. This theorem shows, in a sense, that at most a bounded number of apices are allowed in these classes; and intuitively, splitting an apex with unbounded degree might result in an unbounded number of apices and is thus not allowed in general.

Proposition 3.2. *For every $k \in \mathbb{N}$, there exists a K_6-minor-free graph G_k, so that, if the vertices of G_k are split in any way to achieve a maximum degree of 3, the resulting graph G_k' includes a K_k-minor.*

Proof (*Sketch*). We define G_k to be a sufficiently large planar grid-graph augmented by an apex as follows: consider a sequence S of numbers between 1 and k, so that each possible pair of these k numbers is at least once adjacent in S. Let $t < k^2$ be the length of this sequence. Choose a set W of t vertices in the grid that are sufficiently far away from each other and add an apex v_0 connected to these t vertices. This completes the definition of G_k, which is clearly K_6-minor-free. Now, no matter how we split the vertices of G_k, the apex v_0 will become a path of t vertices, each one connected to exactly one vertex of W. This path imposes an order on the vertices in W. We label the vertices in W according to this order using the sequence S. Let W_i be the set of vertices in W labeled by i ($1 \leq i \leq k$). For each i, construct a tree T_i that connects the vertices

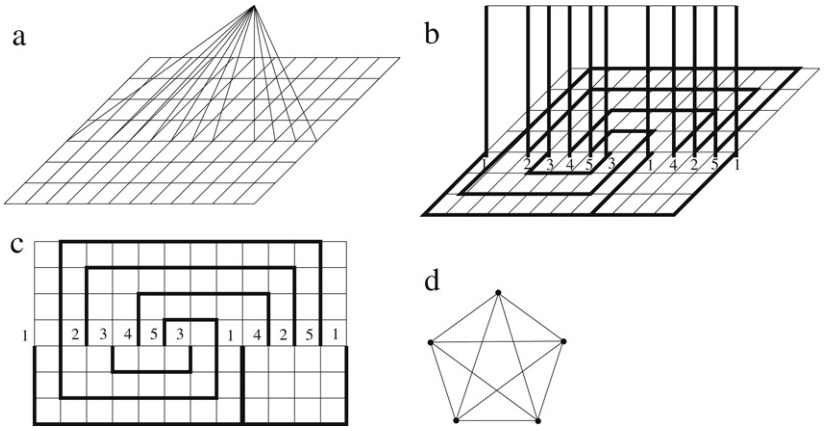

Fig. 1. A simplified example for the proof of Proposition 3.2: the apex of the graph in (a) is split, resulting in the graph (b); the vertices are labeled and connected according to the disjoint trees in (c); contracting the thick edges in (b) results in a K_5-minor (d).

of W_i in the planar grid. Note that if the grid is sufficiently large and the vertices in W are sufficiently far away from each other, it is easily possible to choose the trees T_i to be all disjoint. Let U be the set of edges connecting the vertices in W with the path resulted from splitting v_0. Now, if we contract the trees T_i and the edges in U and delete redundant edges, what remains is a K_k-minor (see Fig. 1). □

3.1. Our generalized recursive division algorithm

Our modified algorithm is given in Algorithm 3.1. Our modifications are only in three places but as we already discussed, they are essential to make the algorithm work for all proper minor-closed graph classes. In this subsection, we discuss the algorithm and our modifications thereof in detail and in the next subsection, we present its proof of correctness.

Algorithm 3.1: Generalized Recursive Division Algorithm

Input : An undirected graph $G = (V, E)$ excluding a K_ℓ-minor.
Output: A recursive division tree T for G satisfying inequality (1) for all r_i's exceeding a constant.

```
begin
    // partition and contract the graph recursively
    let G_0 := G, z_0 := 2, i := 0;
    while the number of vertices in G_i > n / log n do
        let G_{i+1} := H-Partition(G_i, z_i, ℓ);
        let z_{i+1} := 14^{z_i^{1/7}}, i := i + 1;
    let I := i − 1;
    // divide the graphs and build recursive division tree
    let v_G be the root of T;
    let D_{I+1} be the trivial division of G_{I+1} consisting of a single region;
    for i := I downto 0 do
        for each region R of D_{i+1} do
            let S_R be the boundary-vertices of R in the division D_{i+1};
            let D_R := Divide(R, S_R, z_i, ℓ);
            for each region R' of D_R do
                expand R' into a region R'' of G_i by expanding every vertex;
                assign each boundary edge to one of the regions it occurs in;
                create a child v_{R''} of v_R in T;
        let D_i be the decomposition of G_i consisting of the regions R'' above;
    // add the leaves
    for each edge e of each region R of D_0 do
        create a child v_e of v_R in T;
    return T;
end
```

Let the input graph be $G = (V, E)$. In the first phase of the algorithm, the input graph is reduced in size by building minors. Specifically, starting with $G_0 = G$, a sequence of graphs $G_0, G_1, \ldots, G_{l+1}$ is built, so that for $i > j$, G_i is a minor of G_j and G_{l+1} is the first graph in the sequence having less than $n/\log n$ vertices. To this end, a sequence of parameters z_i is used to specify the size of the next graph in the sequence as follows: let n_i be the number of vertices of G_i. In the original algorithm, the sequence is defined as $z_0 = 2$ and $z_{i+1} = 7^{z_i^{1/5}}$ and has the effect that G_i is partitioned into at most n_i/z_i connected components, each one having at most $3z_i$ vertices (using Frederickson's `FindClusters` [9]). Each of these components is contracted to construct G_{i+1}, a minor of G_i with $n_{i+1} \leq n_i/z_i$ vertices.

Our first two changes occur in this phase of the algorithm. First, instead of using Frederickson's `FindClusters`, we make use of the `H-Partition` procedure of Reed and Wood [12], to achieve a similar effect without depending on the graph having bounded degree (cf. Section 2.2). Secondly, we had to change the definition of the z_i's to be $z_0 = 2$ and $z_{i+1} = 14^{z_i^{1/7}}$. This is due the fact that in order to prove inequality (1) in our case, we need the exponent of z_i to be $\frac{1}{7}$ instead of $\frac{1}{5}$; but then, in order to ensure that the z_i's still grow (extremely fast) towards infinity, the base of the exponentiation had to be changed from 7 to 14, too. Indeed, 14 is the smallest integer that can be used, so that the z_i's grow towards infinity. Now, using the `H-Partition` procedure as in Corollary 2.2, we still have that $n_{i+1} \leq n_i/z_i$ but now, each vertex of G_{i+1} represents at most $c_0 z_i$ vertices of G_i (instead of the original $3z_i$).

In the second phase, the algorithm works backwards from G_{l+1} towards G_0, building (r, s)-divisions and a recursive division tree as follows: it starts with the trivial division D_{l+1} of G_{l+1} as a single region and initializes the recursive division tree T with a single node v_G. Then, for each G_i, it considers each region R of G_{i+1} and builds an (r, s)-division on it (with appropriate values of r and s defined below); each resulting region R' of R (of G_{i+1}) is turned into a region R'' of G_i by expanding every vertex into the at most $c_0 z_i$ vertices it represents in G_i; afterwards, a child $v_{R''}$ of v_R is added to T. The division D_i is defined to be the decomposition of G_i by the regions R'' obtained this way. Note that a boundary vertex is expanded in multiple regions, creating multiple copies of the edges it expands to; there should be only one copy of these edges and this may be achieved by assigning them to one of these regions arbitrarily.

It remains to specify how exactly and with what parameters the (r, s)-division is built in the iteration above. This is the third place where our algorithm differs from the original. The original algorithm uses a modified version of Frederickson's (r, s)-division algorithm [10], called `Divide`, as follows: it takes three parameters G, S and r and divides the edges of an n-vertex graph G into at most $c_2(|S|/\sqrt{r} + \frac{n}{r})$ regions, each one having at most r vertices and at most $c_1\sqrt{r}$ boundary vertices, where c_1 and c_2 are constants; a vertex is considered as a boundary vertex if (i) it belongs to more than one region, or (ii) it belongs to the set S. Internally, the linear-time planar $O(\sqrt{n})$-separator of Lipton and Tarjan [11] is used to achieve the desired division. We make use of the linear-time $O(n^{\frac{2}{3}})$-separator of Reed and Wood [12] instead; our `Divide` procedure takes four parameters G, S, r, and ℓ and has the properties specified in Lemma 3.4 below; as it can be seen in the lemma, a number of constants and exponents are changed. The parameter ℓ is a constant taken to indicate the fixed excluded K_ℓ-minor.

In the last phase of the algorithm, the edges of each region R of D_0 are added as children of the node v_R to the recursive division tree T. This completes the description of our generalized algorithm.

3.2. Correctness of our generalized recursive division algorithm

Theorem 3.3. *Algorithm* 3.1 *is a linear-time algorithm that given a K_ℓ-minor-free graph G, finds an $(\bar{r}, f)$-recursive division of G that satisfies inequality (1) for all r_i exceeding a constant and whose recursive division tree has $O(n)$ nodes.*

The proof of the correctness of the algorithm follows the proof of Henziger et al. [8] very closely. For the sake of completeness, and since a number of subtle details and calculations have to be filled in and replaced at several places, we have included the full proof in this section; only the proof of Lemma 3.4 is left for the Appendix. This lemma shows the correctness of the `Divide` procedure and is based on the original proof of Frederickson [10]; Lemmas 3.5–3.7 step-by-step complete the proof of Theorem 3.3.

Lemma 3.4. *Replacing the planar separator in Frederickson's `Divide` procedure* [10] *with the separator algorithm of Reed and Wood* [12] *causes the `Divide` (G, S, r, ℓ) procedure to work as follows (where G is a graph with n vertices and excludes K_ℓ as a minor and c_1 and c_2 are constants depending only on ℓ):*

- *it divides G into at most $c_2(|S|/r^{\frac{2}{3}} + \frac{n}{r})$ regions;*
- *each region has at most r vertices;*
- *each region has at most $c_1 r^{\frac{2}{3}}$ boundary vertices, where the vertices in S also count as boundary;*
- *it takes time $O(n \log n)$.*

Recall that we start with the graph $G_0 = G$ and repeatedly apply the procedure `H-Partition` to each G_i to obtain G_{i+1}. For each i, let n_i denote the number of vertices of G_i. Afterwards we work our way back from G_{l+1} to G_0 and obtain a division D_i on each G_i. Let k_i denote the number of regions of D_i. Note that the recursive division tree T has depth $l + 1$.

The following proof has four parts. First, we show that each region of the division D_i has at most $O(z_i^2)$ vertices and at most $O(z_i^{\frac{5}{3}})$ boundary vertices. Second, we show that the number k_i of regions is $O(n_i/z_i^2)$. Third, we show that Algorithm 3.1 takes linear time and finally, we show that the division fulfills inequality (1).

For notational convenience, let $z_{I+1} = \sqrt{n_{I+1}}$, so the single region of the division D_{I+1} of G_{I+1} has z_{I+1}^2 vertices. Consider iteration $i \leq I$ in the second phase of the algorithm. By the correctness of `Divide`, the decomposition D_R of a region of D_{i+1} consists of regions R' of size at most z_i. By the correctness of `H-Partition`, each vertex of G_{i+1} expands to at most $c_0 z_i$ vertices of G_i. Hence, each region R'' obtained from R' by expanding its vertices has size at most $c_0 z_i^2$. Similarly, each region R' has at most $c_1 z_i^{\frac{2}{3}}$ boundary vertices by the correctness of `Divide`, so the corresponding region R'' has at most $c_0 c_1 z_i^{\frac{5}{3}}$ boundary vertices.

Lemma 3.5. *The number k_i of regions in the division D_i is $O(n_i/z_i^2)$.*

Proof. We show by reverse induction on i that $k_i \leq c_3 n_i/z_i^2$ for all $i \geq i_0$, where i_0 and c_3 are constants to be determined. For the base case, we have $k_{I+1} = 1$.

Consider iteration $i \leq I$ in the second phase, and suppose $i \geq i_0$. The regions of D_i are obtained by subdividing the k_{i+1} regions comprising the division of G_{i+1}. Since $n_{i+1} \leq n_i/z_i$ and $z_{i+1}^2 \geq z_i$, we have by the induction hypothesis that

$$k_{i+1} \leq c_3 n_{i+1}/z_{i+1}^2 \leq c_3 n_i/z_i^2. \tag{2}$$

Each region R of the division of G_{i+1} has $|S_R| \leq c_0 c_1 z_{i+1}^{\frac{5}{3}}$ boundary vertices. Summing over all regions R in D_{i+1}, we obtain

$$\begin{aligned}\sum_R n_R &= \sum_R (\#\text{ of nonboundary vertices } + \#\text{ of boundary vertices})\\ &\leq n_{i+1} + \sum_R c_0 c_1 z_{i+1}^{\frac{5}{3}}\\ &\leq n_{i+1} + c_0 c_1 k_{i+1} z_{i+1}^{\frac{5}{3}}.\end{aligned} \tag{3}$$

For each region R, by correctness of `Divide`, the number of subregions into which R is divided is at most $c_2(|S_R|/z_i^{\frac{2}{3}} + n_R/z_i)$, which is in turn at most $c_2(c_0 c_1 z_{i+1}^{\frac{5}{3}}/z_i^{\frac{2}{3}} + n_R/z_i)$. Summing over all such regions R and using (3) and (2), we infer that the total number of subregions is at most

$$\begin{aligned}\sum_R c_2(c_0 c_1 z_{i+1}^{\frac{5}{3}}/z_i^{\frac{2}{3}} + n_R/z_i) &= c_0 c_1 c_2 k_{i+1} z_{i+1}^{\frac{5}{3}}/z_i^{\frac{2}{3}} + c_2 \sum_R n_R/z_i\\ &\leq c_0 c_1 c_2 k_{i+1} z_{i+1}^{\frac{5}{3}}/z_i^{\frac{2}{3}} + c_2(n_{i+1} + c_0 c_1 k_{i+1} z_{i+1}^{\frac{5}{3}})/z_i\\ &\leq c_0 c_1 c_2 \left(\frac{c_3 n_{i+1}}{z_{i+1}^2}\right) z_{i+1}^{\frac{5}{3}}/z_i^{\frac{2}{3}} + c_2 n_{i+1}/z_i + c_0 c_1 c_2 \left(\frac{c_3 n_{i+1}}{z_{i+1}^2}\right) z_{i+1}^{\frac{5}{3}}/z_i\\ &\leq c_0 c_1 c_2 c_3 n_{i+1}/(z_i^{\frac{2}{3}} z_{i+1}^{\frac{1}{3}}) + c_2 n_{i+1}/z_i + c_0 c_1 c_2 c_3 n_{i+1}/(z_i z_{i+1}^{\frac{1}{3}})\\ &\leq c_0 c_1 c_2 c_3 n_i/(z_i^{\frac{5}{3}} z_{i+1}^{\frac{1}{3}}) + c_2 n_i/z_i^2 + c_0 c_1 c_2 c_3 n_i/(z_i^2 z_{i+1}^{\frac{1}{3}})\end{aligned} \tag{4}$$

where in the last line we use the fact that $n_{i+1} \leq n_i/z_i$. We have obtained an upper bound on the total number of subregions into which the regions of D_{i+1} are divided. Each subregion becomes a region of D_i. Thus, we have in fact bounded k_i, the number of regions of D_i. To complete the induction step, we show that each of the three terms in (4) is bounded by $c_3 n_i/3z_i^2$.

The second term, $c_2 n_i/z_i^2$, is bounded by $c_3 n_i/3z_i^2$ if we choose $c_3 \geq 3c_2$. The third term is smaller than the first term. As for the first term, recall that $z_{i+1} = 14^{z_i^{1/7}}$. For sufficiently large choice of i_0, we can ensure that $i \geq i_0$ implies $z_{i+1}^{\frac{1}{3}} \geq 3c_0 c_1 c_2/z_i^{\frac{2}{3}}$. Thus, the first term is also bounded as desired.

We conclude that $k_i \leq c_3 n_i/z_i^2$, completing the induction step. We have shown this inequality holds for all $i \geq i_0$. As for $i < i_0$, clearly $k_i \leq (z_i^2) n_i/z_i^2 \leq (z_{i_0}^2) n_i/z_i^2$. Thus, by choosing c_3 to exceed the constant $z_{i_0}^2$, we obtain the lemma for every i. □

Lemma 3.6. *The algorithm runs in linear time.*

Proof. The time required to form the graphs $G_1, G_2, \ldots, G_{I+1}$ is $O(\sum_i n/z_i)$, which is $O(n)$. For $i \leq I$, the time to apply `Divide` to a region R of G_{i+1} with n_R vertices is $O(n_R \log n_R)$. Each such region has $O(z_{i+1}^2)$ vertices, so the time is $O(n_R \log z_{i+1})$. Summed over all regions R, we get $\sum_R O(n_R \log z_{i+1}) = O(n_{i+1} \log z_{i+1})$. The time to obtain the induced division of G_i is $O(n_i)$.

Thus, the time to obtain divisions of all the G_i's is $\sum_i O(n_{i+1} \log z_{i+1})$. Since $n_{i+1} \leq n_i/z_i \leq n/z_i$ and $\log z_{i+1} = O(z_i^{\frac{1}{7}})$, the sum is $O(n)$. □

Lemma 3.7. *The recursive division obtained by Algorithm* 3.1 *satisfies inequality* (1).

Proof. First, note that combining the inequalities $n_{i+1} \leq n_i/z_i$, we obtain

$$n_i \leq n/\prod_{j<i} z_j. \tag{5}$$

Note moreover that each vertex of G_i expands to at most $\prod_{j<i} c_0 z_j$ vertices of G.

Consider the division D_i of G_i, and the division it induces on G. The division D_i consists of $O(n_i/z_i^2)$ regions, each having $O(z_i^2)$ vertices and $O(z_i^{\frac{5}{3}})$ boundary vertices. This induces $O(n_i/z_i^2)$ regions in G, each consisting of $O(z_i^2 \prod_{j<i} c_0 z_j)$ vertices and $O(z_i^{\frac{5}{3}} \prod_{j<i} c_0 z_j)$ boundary vertices.

Let $r_i = z_i^2 \prod_{j<i} z_j$ and define

$$f(r_i) = z_i^{\frac{5}{3}} \prod_{j<i} c_0 z_j. \tag{6}$$

Then, by (5), the induced division of G has $O(n/r_i)$ regions each with $O(r_i c_0^i)$ vertices and $O(f(r_i))$ boundary vertices. Since $c_0^i = O(\prod_{j\leq i} z_j)$, we get that the number of vertices per region is $O(r_i^2)$. We have

$$\frac{r_i}{f(r_i)} = \frac{z_i^2}{z_i^{\frac{5}{3}} c_0^{i-1}} = \frac{z_i^{\frac{1}{3}}}{c_0^{i-1}}. \tag{7}$$

Using the definition of z_i, one can verify that $z_{i-1} = \theta(\log^7 z_i)$ and $\prod_{j<i} z_j = O(\log^8 z_i)$. Hence

$$f(r_{i-1}) = c_0^{i-2} z_{i-1}^{\frac{5}{3}} \prod_{j<i-1} z_j = c_0^{i-2} O(\log^{\frac{35}{3}} z_i \log^8 \log z_i). \tag{8}$$

We also have

$$\log r_{i+1} = \log\left(z_{i+1}^2 \prod_{j\leq i} z_j\right) = O(\log(z_{i+1}^2 \log^8 z_{i+1})) = O(\log z_{i+1}) = O(z_i^{\frac{1}{7}}) \tag{9}$$

and consequently $\sum_{j=1}^{i+1} \log r_j = O(z_i^{\frac{1}{7}})$. For a sufficiently large constant i_0, we have for all $i \geq i_0$,

$$\begin{aligned} 8^i f(r_{i-1}) \log r_{i+1} \left(\sum_{j=1}^{i+1} \log r_j\right) &\leq 8^i c_0^{i-2} O(\log^{\frac{35}{3}} z_i \log^8 z_i) O(z_i^{\frac{1}{7}}) O(z_i^{\frac{1}{7}}) \\ &= 8^i c_0^{i-2} O(z_i^{\frac{2}{7}} \log^{20} z_i) \\ &\leq \frac{z_i^{\frac{1}{3}}}{c_0^{i-1}} = \frac{r_i}{f(r_i)}, \end{aligned} \tag{10}$$

since the z_i's grow much faster than any exponential function having a constant in the base; specifically, we can see below that $z_i^{\frac{1}{21}} \geq g_0^i \log^{20} z_i$ for any constant $g_0 \geq 0$ if i is larger than a constant:

$$\begin{aligned} \frac{1}{21} \log z_i \geq i \log g_0 + 20 \log\log z_i &\Leftrightarrow \frac{1}{21} \log 14^{z_i^{\frac{1}{7}}} \geq i \log g_0 + 20 \log\log 14^{z_{i-1}^{\frac{1}{7}}} \\ &\Leftrightarrow z_{i-1}^{\frac{1}{7}} \geq g_1 i + g_2 \log z_{i-1}^{\frac{1}{7}} + g_3, \end{aligned} \tag{11}$$

for some constants g_1, g_2, and g_3. And the last inequality is true if i is large enough, since $z_i^{\frac{1}{7}}$ grows much faster than i. So, inequality (1) is fulfilled for all r_i exceeding the constant r_{i_0}. □

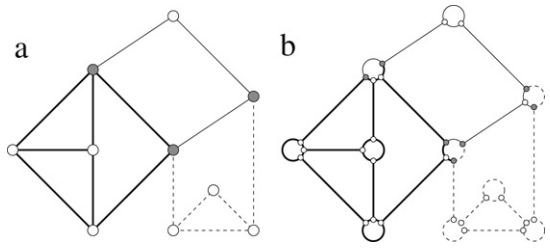

Fig. 2. (a) A given graph with 3 regions indicated by different line-styles and shaded boundary vertices; (b) the transformed graph, such that every vertex has degree at most 3; the number of boundary vertices of each region has exactly doubled.

3.3. Establishing the degree requirement

After having computed a recursive division, we still have to transform the graph to have maximum degree 3; otherwise, Theorem 2.3 can not be applied, see Section 2.4. We can achieve this, using our recursive division, by the following lemma. Note that according to Proposition 3.2. the resulting graph might not be K_ℓ-minor-free but it will still serve our purpose of finding shortest paths in linear time, since it is now accompanied by a recursive division satisfying inequality (1).

Lemma 3.8. *Let G be an edge-weighted directed graph excluding a fixed minor and let T be a recursive division tree representing an $(\bar{r}, f)$-recursive division of G. Then one can replace every vertex of G with a zero-weight cycle to obtain a graph G' and at the same time modify T into a tree T', so that G' has in-/outdegree at most 2 and T' represents an $(\bar{r}, f)$-recursive division of G'. This modification takes linear time.*

Proof. Recall that the leaves of T represent the edges of G and that internal nodes of T correspond to regions of G, namely, the region induced by all the leaves in the subtree of that node. We modify G and T at the same time. First, for every vertex v of G with degree $d(v)$ (the sum of the indegree and outdegree), we add new vertices $v_1, \ldots, v_{d(v)}$ to G. We do an in-order traversal of T and for every leaf of T representing an edge $e = vw$ of G, we do the following: let e be the ith edge of v and the jth edge of w that we encounter. We change the endpoints of e to be the vertices v_i and w_j and add two new zero-weight edges v_iv_{i+1} and w_jw_{j+1} as siblings of e to T (if $i = d(v)$, we use $v_{d(v)}v_1$ instead; same for w). This way, every vertex v of G is replaced by a zero-weight cycle $(v_1, \ldots, v_{d(v)})$ (see Fig. 2). The original vertices of G will become isolated and can be removed. We call the resulting graph G' and the modified recursive division tree T'. Note that since T has size $O(n)$, this procedure takes only linear time. Also note that we only added new leaves to T and thus, the internal nodes of T and T' correspond one-to-one to each other.

Now consider an internal node q' of T'. It represents a region R' of G' and corresponds to a node q of T, representing a region R of G. R has $r^{O(1)}$ vertices and $O(f(r))$ boundary-vertices. The number of edges of R' is at most three times as large as in R and the number of vertices is proportional to the number of edges of R. But R is a subgraph of G, excludes the same fixed minor and thus, the number of its edges is linear in the number of its vertices. Hence, R' still has $r^{O(1)}$ vertices and edges. Also, since R is represented by the subtree rooted at q, its edges were traversed in order while building T' and G'. So, every vertex v in R is replaced by a path $v_i, v_{i+1}, \ldots, v_j$ with $1 \le i \le j \le d(v)$ in R'. Thus, if v is a boundary vertex of R, then instead, we have v_i and v_j as boundary vertices of R'. So R' has at most twice as many boundary vertices as R, i.e. still $O(f(r))$ (see Fig. 2). So, T' represents an $(\bar{r}, f)$-recursive division of G'. □

Proof of Theorem 3.1. Note that up to the choice of the start- and endvertex inside the zero-weight cycles of G', shortest paths in G and G' correspond one-to-one to each other. G' fulfills all the requirements of Theorem 2.3 and combining this with Theorem 3.3, and Lemma 3.8, we obtain our main theorem, namely, Theorem 3.1. □

4. Steiner tree approximation

We show how to implement Mehlhorn's 2-approximation algorithm for the Steiner tree problem [20] in linear time on proper minor-closed graph classes using the result above and the observation that Mehlhorn's distance network is a minor of the input graph. First, we briefly review Mehlhorn's algorithm and then we present our implementation.

4.1. Overview of Mehlhorn's algorithm

Given an n-vertex graph $G = (V, E)$ with nonnegative edge-weights and a vertex-subset K of terminals, one can determine a Steiner tree of K in G as follows: first, build *Mehlhorn's distance network* $N_D^\star = (K, E_D^\star)$, a special graph defined on the set of terminals, in which every edge corresponds to a path in G. To calculate $N_D^\star$, we first have to partition the graph into *Voronoi regions* with respect to the set of terminals K. Every vertex of the graph belongs to the Voronoi region of its closest terminal (if a vertex happens to have the same distance to more than one terminal, it should belong to the Voronoi region of the terminal with the smallest index). Voronoi regions in graphs can be calculated easily using a shortest-paths

computation: add a super-source s_0 to the graph and connect it to every terminal with a directed zero-weight edge; find the shortest paths from s_0 to every vertex and then remove s_0 from the resulting shortest-paths tree. The tree falls apart into $|K|$ connected components, each having a terminal as their root. These components correspond exactly to the Voronoi regions of the terminals. Using Dijkstra's algorithm, one obtains a running time of $O(n \log n + m)$ for general graphs.

In the distance network $N_D^\star$, there exists an edge between two terminals u and v if and only if there exists an edge between two vertices x and y in G, so that x belongs to the Voronoi region of u and y belongs to the Voronoi region of v. The weight of such an edge is the length of the shortest such paths connecting u and v. Once the Voronoi regions of G with respect to K are determined, $N_D^\star$ can be constructed in linear time using bucket sort.

After the distance network $N_D^\star$ is determined, one can find its minimum spanning tree and replace every edge with the corresponding path in G. Mehlhorn shows that the resulting graph is indeed a tree and its weight is at most $(2 - \frac{2}{|K|})$ times the weight of the minimum Steiner tree of K in G. The implementation he offers runs in time $O(n \log n + m)$ for general graphs.

4.2. A linear-time implementation for proper minor-closed classes

Theorem 4.1. *There is a linear-time algorithm that calculates a* 2*-approximation for the Steiner minimum tree problem in any proper minor-closed class of graphs.*

We first show how to find the Voronoi regions in linear time. In graphs excluding a fixed minor K_ℓ, we observe that the graph with an added super-source will exclude $K_{\ell+1}$; so, Theorem 3.1 applies and shortest paths can be calculated in linear time. Alternatively, using a similar method as in Section 3.3, one can first find a recursive division of G and then add the super-source and its edges to G and to the recursive division. This could result in much better constants in the running time of the algorithm, especially for planar graphs. We get

Lemma 4.2. *For a graph G excluding a fixed minor and having nonnegative edge-weights and a given set of terminals K in G, the Voronoi regions of G with respect to K can be determined in linear time.*

Corollary 4.3. *In a proper minor-closed class of graphs, the distance network $N_D^\star$ can be calculated in linear time for any given set of terminals in a given graph from the class.*

The next step of Mehlhorn's algorithm is to calculate the minimum spanning tree of $N_D^\star$. But notice that $N_D^\star$ is obtained by contracting the Voronoi regions of the graph (which are connected) and removing loops and parallel edges, i.e.

Observation 4.4. *For a given graph G and a set of terminals, the distance network $N_D^\star$ is a minor of G.*

Thus, $N_D^\star$ belongs to the same proper class of minor-closed graphs as G and one can apply the linear-time minimum spanning tree algorithm of Mares [24]. When we are dealing with planar graphs, the algorithm of Cheriton and Tarjan can be used [25]. As mentioned before, the last step of Mehlhorn's algorithm is to replace the edges of $N_D^\star$ with the corresponding paths from G and this can clearly be done in linear time. Hence, Theorem 4.1 is proven. □

Acknowledgement

The first author was supported by the Deutsche Forschungsgemeinschaft (DFG), grant MU1482/3-1.

Appendix. Proof of Lemma 3.4

Lemma A.1. *Replacing the planar separator in Frederickson's* `Divide` *procedure* [10] *with the separator algorithm of Reed and Wood* [12] *causes the* `Divide` (G, S, r, ℓ) *procedure to work as follows (where G is a graph with n vertices and excludes K_ℓ as a minor and c_1 and c_2 are constants depending only on ℓ):*

- *it divides G into at most $c_2(|S|/r^{\frac{2}{3}} + \frac{n}{r})$ regions;*
- *each region has at most r vertices;*
- *each region has at most $c_1 r^{\frac{2}{3}}$ boundary vertices, where the vertices in S also count as boundary;*
- *it takes time $O(n \log n)$.*

Proof. In the following, when we refer to boundary vertices, we mean vertices that belong to more than one region or vertices that belong to the set S. The `Divide` procedure works as follows: assign weight $\frac{1}{n}$ to each vertex of G and find a $O(n^{\frac{2}{3}})$-separator in G; recursively apply the separation algorithm to each region with more than r vertices. Now each region has at most r vertices. While there is a region with more than $c_1 r^{\frac{2}{3}}$ boundary vertices, do the following: if such a region has n' boundary vertices, assign weight $\frac{1}{n'}$ to each of them, assign weight zero to the other vertices of that region and apply the

separator theorem. In the end, all regions will have the desired properties and the algorithm takes time $O(n \log n)$. It remains to show the bound on the number of the regions.

Consider the division before the regions are further split to enforce the requirement on the number of boundary vertices (i.e. just when we have achieved that each region has size at most r). Let V_B be the set of vertices that are included in more than one region. For a vertex $v \in V_B$, let $b(v)$ be one less than the number of regions that contain v in the division. Let $B(n, r)$ be the total of $b(v)$ over all vertices $v \in V_B$. Thus, $B(n, r)$ is the sum of the number of vertices $v \in V_B$ weighted by the count $b(v)$. From the separation theorem in [12], we have the following recurrence:

$$\begin{aligned} &B(n, r) \leq d_0 n^{\frac{2}{3}} + B(\alpha n, r) + B((1-\alpha)n, r) \quad \text{for } n > r, \\ &B(n, r) = 0 \quad \text{for } n \leq r \end{aligned} \tag{A.1}$$

where d_0 is a constant and $\frac{1}{2} \leq \alpha \leq \frac{2}{3}$. We claim that

$$B(n, r) \leq d_1 \frac{n}{r^{\frac{1}{3}}} - d_2 n^{\frac{2}{3}} \quad \text{for } n \geq \frac{r}{3}, \tag{A.2}$$

with some constants d_1 and d_2. The claim can be shown by induction:

As the base of the induction, we consider the cases $\frac{r}{3} \leq n \leq r$. Note that since after splitting a region, each subregion still has at least one-third of the total vertices, it is sufficient to only consider graphs with at least $r/3$ vertices. By choosing $d_1 \geq 3^{\frac{1}{3}} d_2$, we have

$$\frac{d_1 n}{r^{\frac{1}{3}}} \geq \frac{3^{\frac{1}{3}} d_2 n}{3^{\frac{1}{3}} n^{\frac{1}{3}}} = d_2 n^{\frac{2}{3}} \Rightarrow \frac{d_1 n}{r^{\frac{1}{3}}} - d_2 n^{\frac{2}{3}} \geq 0 = B(n, r). \tag{A.3}$$

For the inductive step, i.e. for $n > r$, we have

$$\begin{aligned} B(n, r) &\leq d_0 n^{\frac{2}{3}} + d_1 \frac{\alpha n}{r^{\frac{1}{3}}} - d_2 \alpha^{\frac{2}{3}} n^{\frac{2}{3}} + d_1 \frac{(1-\alpha)n}{r^{\frac{1}{3}}} - d_2 (1-\alpha)^{\frac{2}{3}} n^{\frac{2}{3}} \\ &= d_1 \frac{n}{r^{\frac{1}{3}}} + n^{\frac{2}{3}} (d_0 - d_2 \alpha^{\frac{2}{3}} - d_2 (1-\alpha)^{\frac{2}{3}}) \\ &\leq d_1 \frac{n}{r^{\frac{1}{3}}} - d_2 n^{\frac{2}{3}} \end{aligned} \tag{A.4}$$

if we choose $d_2 \leq d_2 \alpha^{\frac{2}{3}} + d_2 (1-\alpha)^{\frac{2}{3}} - d_0$. This can be achieved by setting $d_2 = 5d_0 \geq \frac{d_0}{\alpha^{\frac{2}{3}} + (1-\alpha)^{\frac{2}{3}} - 1}$.

In particular, we have shown so far that $B(n, r) = O(n/r^{\frac{1}{3}})$. The sum of the number of vertices in each region is $n + B(n, r) = n + O(n/r^{\frac{1}{3}})$ and each region has $\Theta(r)$ vertices, so the number of regions we have so far is $\Theta(n/r)$.

Let t_i be the number of regions with i boundary vertices (recall that in our definition, the set of boundary vertices is $V_B \cup S$). We have

$$\sum_i i t_i = \sum_{v \in V_B} (b(v) + 1) + |S \setminus V_B| < 2B(n, r) + |S| = O(n/r^{\frac{1}{3}} + |S|). \tag{A.5}$$

Let $s(i)$ be an upper bound on the number of splits that have to be applied to a graph with at most r vertices and i boundary vertices, until each of its regions has at most $c_1 r^{\frac{2}{3}}$ boundary vertices, for a constant c_1 to be determined. We have that

$$\begin{aligned} &s(i) \leq s(\alpha i + d_0 r^{\frac{2}{3}}) + s((1-\alpha)i + d_0 r^{\frac{2}{3}}) + 1 \quad \text{for } i > c_1 r^{\frac{2}{3}} \\ &s(i) = 0 \quad \text{for } i \leq c_1 r^{\frac{2}{3}} \end{aligned} \tag{A.6}$$

where $\frac{1}{2} \leq \alpha \leq \frac{2}{3}$. We claim that

$$s(i) \leq \frac{d_3 i}{c_1 r^{\frac{2}{3}}} - \frac{2 d_0 d_3}{c_1} - 1 \quad \text{for } i \geq \frac{c_1 r^{\frac{2}{3}}}{3} \tag{A.7}$$

for some constant d_3. We prove our claim by induction. Like in the previous induction, for the base case we may assume $\frac{c_1 r^{\frac{2}{3}}}{3} \leq i \leq c_1 r^{\frac{2}{3}}$. By choosing $d_3 = 12$ and $c_1 = 8d_0$, we have

$$\frac{d_3 i}{c_1 r^{\frac{2}{3}}} - \frac{2 d_0 d_3}{c_1} - 1 \geq 12 \cdot \frac{1}{3} - \frac{24 d_0}{8 d_0} - 1 = 0 = s(i). \tag{A.8}$$

For the inductive step with $i > c_1 r^{\frac{2}{3}}$, note that $\alpha i + d_0 r^{\frac{2}{3}} \leq \frac{2}{3} i + \frac{d_0}{c_1} i \leq (\frac{2}{3} + \frac{1}{8}) i < i$. The same way, we have $(1-\alpha) i + d_0 r^{\frac{2}{3}} < i$. So, we may apply the induction hypothesis to (A.6) and a straightforward calculation will prove our claim.

We have shown that for a region with i boundary vertices, where $i > c_1 r^{\frac{2}{3}}$, at most $\frac{d_3 i}{c_1 r^{\frac{2}{3}}}$ splits need be done for some constants c_1 and d_3. This will result in at most $d_0 r^{\frac{2}{3}}$ new boundary vertices per split and a total of at most $d_3 i / (c_1 r^{\frac{2}{3}})$ new regions. Thus, the total number of new boundary vertices is at most

$$\sum_i (d_0 r^{\frac{2}{3}})(d_3 i / (c_1 r^{\frac{2}{3}})) t_i \leq \frac{d_0 d_3}{c_1} \sum_i i t_i = O(n/r^{\frac{1}{3}} + |S|). \tag{A.9}$$

The number of new regions is at most

$$\sum_i (d_3 i / (c_1 r^{\frac{2}{3}})) t_i = \frac{d_3}{c_1 r^{\frac{2}{3}}} O(n/r^{\frac{1}{3}} + |S|) = O(n/r + |S|/r^{\frac{2}{3}}). \quad \square \tag{A.10}$$

References

[1] E. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik 1 (1959) 269–271.
[2] M.L. Fredman, R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, Journal of the ACM 34 (3) (1987) 596–615.
[3] S. Pettie, V. Ramachandran, A shortest path algorithm for real-weighted undirected graphs, SIAM Journal on Computing 34 (6) (2005) 1398–1431.
[4] A.V. Goldberg, A simple shortest path algorithm with linear average time, in: ESA'01: Proceedings of the 9th Annual European Symposium on Algorithms, in: Lecture Notes in Computer Science, vol. 2161, Springer-Verlag, London, UK, 2001, pp. 230–241.
[5] M. Thorup, Undirected single-source shortest paths with positive integer weights in linear time, Journal of the ACM 46 (3) (1999) 362–394.
[6] T. Hagerup, Improved shortest paths on the word RAM, in: U. Montanari, J.D.P. Rolim, E. Welzl (Eds.), ICALP, in: Lecture Notes in Computer Science, vol. 1853, Springer, 2000, pp. 61–72.
[7] U. Zwick, Exact and approximate distances in graphs — A survey, in: ESA'01: Proceedings of the 9th Annual European Symposium on Algorithms, in: Lecture Notes in Computer Science, vol. 2161, Springer-Verlag, London, UK, 2001, pp. 33–48.
[8] M.R. Henzinger, P.N. Klein, S. Rao, S. Subramanian, Faster shortest-path algorithms for planar graphs, Journal of Computer and System Sciences 55 (1) (1997) 3–23. Previously appeared in the Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, STOC'94, ACM Press, pp. 27–37.
[9] G.N. Frederickson, Data structures for on-line updating of minimum spanning trees, with applications, SIAM Journal on Computing 14 (4) (1985) 781–798.
[10] G.N. Frederickson, Fast algorithms for shortest paths in planar graphs, with applications, SIAM Journal on Computing 16 (6) (1987) 1004–1022.
[11] R.J. Lipton, R.E. Tarjan, A separator theorem for planar graphs, SIAM Journal on Applied Mathematics 36 (2) (1979) 177–189.
[12] B. Reed, D.R. Wood, Fast separation in a graph with an excluded minor, in: S. Felsner (Ed.), 2005 European Conference on Combinatorics, Graph Theory and Applications, EuroComb'05, in: DMTCS Proceedings, Discrete Mathematics and Theoretical Computer Science, vol. AE, 2005, pp. 45–50.
[13] N. Alon, P. Seymour, R. Thomas, A separator theorem for nonplanar graphs, Journal of the American Mathematical Society 3 (4) (1990) 801–808.
[14] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller, J.W. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, 1972, pp. 85–103.
[15] M. Bern, P. Plassmann, The Steiner problem with edge lengths 1 and 2, Information Processing Letters 32 (4) (1989) 171–176.
[16] M. Chlebík, J. Chlebíková, Approximation hardness of the Steiner tree problem on graphs, in: M. Penttonen, E.M. Schmidt (Eds.), SWAT, in: Lecture Notes in Computer Science, vol. 2368, Springer, 2002, pp. 170–179.
[17] G. Robins, A. Zelikovsky, Improved Steiner tree approximation in graphs, in: Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, 2000, pp. 770–779.
[18] E.-A. Choukhmane, Une heuristique pour le problème de l'arbre de Steiner, Recherche Opèrationelle. 12 (1978) 207–212.
[19] J. Plesnik, A bound for the Steiner problem in graphs, Mathematica Slovaca 31 (1981) 155–163.
[20] K. Mehlhorn, A faster approximation algorithm for the Steiner problem in graphs, Information Processing Letters 27 (1988) 125–128.
[21] M. Garey, D. Johnson, The rectilinear Steiner tree problem is $\mathcal{NP}$-complete, SIAM Journal on Applied Mathematics 32 (1977) 826–834.
[22] G. Borradaile, C. Kenyon-Mathieu, P.N. Klein, A polynomial-time approximation scheme for Steiner tree in planar graphs, in: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, 2007, pp. 1285–1294.
[23] G. Borradaile, C. Kenyon-Mathieu, P.N. Klein, Steiner tree in planar graphs: An $O(n \log n)$ approximation scheme with singly exponential dependence on epsilon, in: Proceedings of the 10th Workshop on Algorithms and Data Structures, in: Lecture Notes in Computer Science, vol. 4619, Springer, 2007, pp. 275–286.
[24] M. Mares, Two linear time algorithms for MST on minor closed graph classes, Archivum Mathematicum 40 (3) (2004) 315–320.
[25] D.R. Cheriton, R.E. Tarjan, Finding minimum spanning trees, SIAM Journal on Computing 5 (4) (1976) 724–742.
[26] N. Robertson, P.D. Seymour, Graph minors. XX. Wagner's conjecture, Journal of Combinatorial Theory Series B 92 (2) (2004) 325–357.
[27] W. Mader, Homomorphieeigenschaften und mittlere Kantendichte von Graphen, Mathematische Annalen 174 (1967) 265–268.
[28] N. Robertson, P. Seymour, Graph minors. XVI. Excluding a non-planar graph, Journal of Combinatorial Theory Series B 89 (1) (2003) 43–76.