An Optimal Parallel Algorithm for Minimum Spanning Trees in Planar Graphs

Ka Wong Chong[1] and Christos Zaroliagis[2,3](✉)

[1] Department of Computer Science, The University of Hong-Kong, Porfulam Road, Porfulam, Hong Kong
[2] Department of Computer Engineering and Informatics, University of Patras, 26504 Patras, Greece
zaro@ceid.upatras.gr
[3] Computer Technology Institute and Press "Diophantus", N. Kazantzaki Str., Patras University Campus, 26504 Patras, Greece

Abstract. We present an optimal deterministic $O(n)$-work parallel algorithm for finding a minimum spanning tree on an n-vertex planar graph. The algorithm runs in $O(\log n)$ time on a CRCW PRAM and in $O(\log n \log^* n)$ time on an EREW PRAM. Our results hold for any sparse graph that is closed under taking of minors, as well as for a class of graphs with non-bounded genus.

1 Introduction

The minimum spanning tree problem is one of the most fundamental problems in network optimization with a wealth of theoretical and practical applications (see e.g., [1]). Given a connected n-vertex, m-edge undirected graph G with real edge weights, the *minimum spanning tree* (MST) problem is to find a spanning tree of minimum total weight among all spanning trees of G. The problem has been extensively studied both in sequential and in parallel computation.

In sequential computation, the MST problem has started being investigated as early as 1926 [4]. The currently best deterministic sequential algorithms [5,15, 27] run in almost linear time. The first two [5,15] run on the (classical) unit-cost random access machine (RAM) model of computation, where the only operations allowed on the edge weights are binary comparisons, while the third one [27] is optimal and runs on the pointer machine. Better, linear-time algorithms are known if randomization is allowed [25], or if the input graph is planar [6], or if more powerful models of computation are used [14].

In parallel computation, the MST problem has been studied in the parallel random access machine (PRAM) model of computation, the parallel version of the unit-cost RAM (for more on PRAMs see e.g., [21,26]). In the parallel context, there is a close relationship between the connected components and the MST problem in the sense that almost all parallel algorithms for either of the problems use the *hook-and-contract* approach (also known as Sollin's or Boruvka's approach): initially each vertex represents a component by itself. Then, every

C. Zaroliagis et al. (Eds.): Spirakis Festschrift, LNCS 9295, pp. 169–182, 2015.
DOI: 10.1007/978-3-319-24024-4_11

component C_i hooks to another component C_j by selecting an edge whose one endpoint is in C_i and the other in C_j. After hooking, every (new) component is contracted into a single vertex. The hooking and contraction steps are repeated until there are no more edges connecting different components. While in computing connected components the selection of the hooking edge can be arbitrary, in the MST problem this selection is critical: it has to be the edge with minimum weight. This particular difficulty usually increases the running time and/or the work of the MST algorithms. Further, it has been shown in [24] that the MST problem can be reduced to a connected components problem without an increase in the time; however, the number of processors used (and hence the work) is increased to $m^{1+\varepsilon}$, for some $\varepsilon > 0$.

The results of [8,23] for the MST problem on the EREW PRAM came therefore as a surprise since they matched the corresponding connected components bounds in [9,22]. Namely, in [23] an algorithm for the MST problem is presented running in $O(\log^{3/2} n)$ time and performing $O(m \log^{3/2} n)$ work. In [8] this result is improved to $O(\log n \log\log n)$ time and $O(m \log n \log\log n)$ work. Note that both MST results used much different techniques from those used in the corresponding connected components algorithms. The time for MST (and connected components) on the EREW PRAM was ultimately reduced to $O(\log n)$ in a breakthrough result [10] that used a new technique based on concurrent threads. The algorithm in [10] performs $O((n+m)\log n)$ work.

On the CRCW PRAM, there is still a certain gap in the work performed between the best deterministic connectivity algorithm [13] and the best MST algorithm [29]. The connected components algorithm in [13] runs in $O(\log n)$ time and performs $O(m\alpha(m,n))$ work on an Arbitrary CRCW PRAM. The MST algorithm in [29] runs in $O(\log n)$ time and performs $O(m \log n)$ work on a Common CRCW PRAM. Previous approaches for the MST problem [2,13] achieve similar bounds but on the much stronger Priority CRCW PRAM model.

Optimal-work parallel MST algorithms are known only for the case where randomization is allowed, or for the case of special classes of graphs. Regarding the former, a randomized algorithm is presented in [11] that runs in $O(\log n)$ time and performs $O(m)$ work on an Arbitrary CRCW PRAM, while an EREW PRAM algorithm with the same bounds was presented in [28]. Regarding the latter, for very dense graphs (i.e., $m = \Omega(n^2)$) a CREW PRAM algorithm was given in [7] that runs in $O(\log^2 n)$ time and performs $O(n^2)$ work. For the case of planar graphs, $O(n)$-work deterministic parallel algorithms running in $O(\log n \log^* n)$ time on an EREW PRAM and in $O(\log n)$ time on a CRCW PRAM were given in [18].

In this paper, we present another optimal deterministic parallel algorithm that solves the MST problem in the important case of planar graphs. Our algorithm runs in $O(\log n \log^* n)$ time on an EREW PRAM, or in $O(\log n)$ time on an Arbitrary CRCW PRAM, and performs $O(n)$ work. Our algorithm matches the bounds in [18] as well as those of the best parallel algorithm for computing connected components on the same models of computation and for the same classes of graphs [17]. Our algorithm uses different techniques compared

to those in [17,18] and might constitute a simpler alternative to those algorithms. In addition, our results hold for any sparse graph that is closed under taking of minors, as well as for a class of graphs with non-bounded genus.

The main idea of our algorithm is the following. We perform a number of iterations (as in the hook-and-contract approach), but we maintain the property that the graph we are dealing with has constant degree. However, after a contraction the maximum degree of a graph may increase and hence after a few iterations is no longer bounded by a constant. To overcome this problem, we *expand* the graph into a new one with maximum degree 3 (Sect. 2). The expansion is done such that an MST of the original graph can be easily found from an MST of the expanded graph. Moreover, we can guarantee that for the graphs considered the "contraction rate" is larger than the "expansion rate" so that the algorithm terminates after a logarithmic number of iterations. For simplicity, we present first our algorithm for the case of planar graphs (Sects. 3 and 4). Later (Sect. 5) we discuss how it extends to any class of sparse graphs that is closed under taking of minors, as well as for a class of non-bounded genus graphs.

2 Preliminaries

All graphs throughout the paper are undirected and are assumed to be given in its adjacency list representation. Let $G = (V, E)$ be a connected graph, where $|V| = n$ and $|E| = m$. Let also $w(\cdot)$ be a weight function on the edges of G and let $deg_G(v)$ denote the degree of v in G. If $\forall v \in V$, $deg_G(v) \leq \delta$, then we shall call G a *degree-δ* graph. For any spanning tree T of G, we define its weight, $w(T)$, as the sum of the weights of all the edges in T. Then, a minimum spanning tree of G, denoted by T^*_G, is the one with the minimum weight. Throughout the paper we shall not distinguish between a spanning tree T and its set of edges (unless stated otherwise). To simplify our discussion, we make the following assumptions concerning the minimum spanning tree problem.

A1. No two edges in G have the same weight and consequently T^*_G is unique. We can easily fulfill this assumption by considering the triple $\langle w(e), u, v\rangle$ as the weight of the edge $e = (u, v)$ in the adjacency list of u and compare edge weights using the lexicographic order.

A2. The weight function takes values on the positive reals, i.e., $w : E \rightarrow I\!R^+$. This is not a restriction, since we can always add to all edge weights in G a sufficiently large number $L > 0$ to make them positive. Moreover, it is easy to verify that the MST, say T^+, found in this case is isomorphic to T^*_G and that $w(T^*_G) = w(T^+) - (n-1)L$.

Let $G' = (V', E')$ be a connected subgraph of G, where $V' \subseteq V$ and $E' \subseteq E$. We call the edges in E' *internal* edges of G', and the edges of G with only one endpoint in V' *external* edges of G'. Throughout the paper, the *contraction of G' into a single vertex* is an operation defined as follows: first, remove all internal edges in G' and all but one vertices in V' (the remaining vertex is the vertex representing G' in the contracted graph). Then, replace all multiple edges

that may have been created with the one of minimum weight. (The latter step guarantees that the contracted graph is a simple graph.)

Let $F \subset E$ be a subset of edges of G. Consider the subgraphs of G induced by the edges in F. If every such induced subgraph is a tree, then we say the F *induces a forest in* G. If a vertex $v \in V$ has no edge from F incident on it, then v induces a tree by itself.

We shall need the following well-known property of MSTs (for a proof see e.g., [1]).

Lemma 1. *Let F be a set of edges of G such that F induces a forest in G and $F \subseteq T^*_G$. Let (u, v) be the external edge of minimum weight of a tree in F. Then, a minimum spanning tree of G contains the edge (u, v) and all edges in F.*

The following properties of a planar graph will be useful later.

Lemma 2. *Let G be a weighted planar graph and let F be a set of edges of G that induce a forest in G. Then:*
(i) The number of edges m in G is no more than $3n - 6$.
(ii) If we contract each tree (connected component) in F into a single vertex, the contracted graph G' is still planar.
(iii) If $F \subseteq T^*_G$, then $T^*_G = T^*_{G'} \cup F$.

Properties (i) and (ii), in the above lemma, are well-known properties of planar graphs (see e.g., [20]). Property (iii) follows by Lemma 1.

The following concept plays a key role in our algorithm.

Definition 1. *Let H be a weighted graph and let S be a subset of edges of H that induce a forest. Then, S is said to be a* $(c, f(c))$**-connector** *of H if every edge of S belongs to T^*_H and for each tree T induced by S in H, $c \leq |T| \leq f(c)$, where $|T|$ denotes the number of vertices in T and $f(c)$ is a function of c. By convention, if H has less than c vertices, then $S = T^*_H$.*

In all applications of the above definition, throughout the paper, c will always be a constant. As we shall see in Sect. 4, a $(c, f(c))$-connector S, for constant c, can be computed very efficiently. After contracting the trees induced by S, the graph H is contracted by a factor of at least c. Since further each tree of S contains at most $f(c)$ vertices, the contraction can be done in $O(1)$ time, resulting in a (new) contracted graph H'. The remaining of the edges of T^*_H can now be found in H' (according to Lemma 2 (iii)).

During the execution of our algorithm, we want to maintain the invariant that the graph in processing satisfies the property that every vertex has degree bounded by a fixed constant. However, it can be easily verified that after contracting a graph the maximum degree may increase (i.e., the maximum degree of H' may be larger than that of H). We deal with this problem by expanding the graph. In the following, we describe a simple transformation that implements the expansion. More precisely, the transformation takes as input a graph G and outputs a graph H in which every vertex has degree bounded by 3. Moreover,

the minimum spanning tree of H naturally defines the minimum spanning tree of G (see Lemma 5 below).

The expansion transformation is defined as follows. For every vertex v of G, if $deg_G(v) \leq 3$, then include v and all its edges into H directly. Otherwise, if $deg_G(v) > 3$, we create $t = deg_G(v) - 2$ new vertices, $v_1, v_2, \ldots, v_t$, in H. (We alternatively say that v is *split* into t vertices.) Let $e_1, e_2, \ldots, e_{t+1}, e_{t+2}$ be the edges incident on v in G. Then in H, make e_1 and e_2 incident on v_1, make e_i incident on v_{i-1}, for $3 \leq i \leq t$, and make e_{t+1} and e_{t+2} incident on v_t. Finally, add edges between v_i and v_{i+1}, for $1 \leq i \leq t-1$. Each of these new edges is associated with zero weight. The expansion transformation is illustrated in Fig. 1.

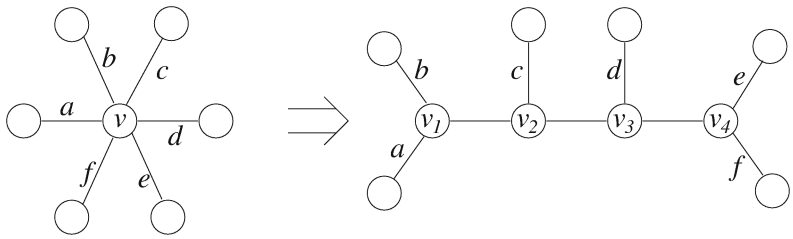

Fig. 1. A vertex v with degree 6, incident on edges a, b, c, d, e, and f, is transformed into four vertices, v_1, v_2, v_3, and v_4. The three new edges $(v_1, v_2), (v_2, v_3)$, and (v_3, v_4) are associated with zero weight.

Lemma 3. *(i) The transformation of a graph G into a degree-3 graph H can be done in $O(\log d)$ time and $O(m)$ work on an EREW PRAM, where d is the maximum degree of a vertex in G.*
(ii) If G is planar, then H is also planar and $n_h \leq 5n - 12$, where n_h is the number of vertices in H.

Proof. (i) It can be easily done using list ranking and segmented parallel prefix computations [21].

(ii) The planarity of H is obvious. For the number of vertices in H, we have that $n_h \leq \sum_{v \in G} \max\{deg_G(v) - 2, 1\} \leq 2m - 2n + n = 2m - n$. As $m \leq 3n - 6$ (Lemma 2 (i)), we have $n_h \leq 5n - 12$. □

The degree-3 graph H, resulting from this transformation, has some useful properties which we discuss next and which allow us to find easily the MST of G, if the MST of H is given.

Lemma 4. *Every edge in H with zero weight belongs to T^*_H.*

Proof. Consider a greedy (e.g., Kruskal's) algorithm that finds T^*_H by selecting edges in non-decreasing order of weight, and discards an edge if it creates a cycle with already selected edges. (The correctness of this approach can be easily verified using Lemma 1.) Now, all zero weight edges must be present in T^*_H, since their weight is smaller than those of the remaining edges in H and they do not form any cycle. □

Lemma 5. *The edges with non-zero weight in T_H^* are the edges of T_G^*.*

Proof. It suffices to prove that: (a) all the non-zero weight edges of T_H^* induce a spanning tree in G; and (b) this spanning tree is the MST of G.

To prove claim (a), first observe that in H there are $n_h - n$ edges which have zero weight. Let X be the non-zero weight edges of T_H^*. By Lemma 4, all the zero weight edges of H must be in T_H^*. Therefore, $|X| = n_h - 1 - (n_h - n) = n - 1$.

Suppose on the contrary that X does not induce a spanning tree in G. Then, there must be a simple cycle $C \subseteq X$ induced by X. Let $u_0, u_1, \ldots, u_{k-1}$ be the vertices of C and $e_0, e_1, \ldots, e_{k-1}$ be its edges, where u_i, $0 \leq i \leq k-1$, is incident on edges $e_{(i-1) \mod k}$ and $e_{i \mod k}$. By the transformation, each u_i will be split into several vertices which are connected through (a path consisting of) the new edges with zero weight. As a result, the edges $e_{(i-1) \mod k}$ and $e_{i \mod k}$ are connected through the zero weight edges in T_H^*. Thus, there is a cycle in H involving the edges of C and the zero weight edges of H. Since by Lemma 4 all zero weight edges of H must be present in T_H^*, we have that C is a cycle in H induced by edges of T_H^*. But this contradicts the fact that T_H^* is a spanning tree of H. Hence, claim (a) is proved.

We now turn to claim (b). Let T_G be the spanning tree in G induced by the non-zero weight edges of T_H^*. Clearly, $w(T_G) = w(T_H^*)$.

Suppose on the contrary that T_G is not the minimum spanning tree and let T_G' be the MST in G. Then, $w(T_G') < w(T_G)$. By the transformation, T_G' defines a spanning tree T_H in H consisting of all the edges of T_G' and all the zero weight edges of H. But then $w(T_H) = w(T_G') < w(T_G) = w(T_H^*)$, a contradiction to the assumption that T_H^* is the MST of H. This ends the proof of claim (b) and the proof of the lemma. □

3 The Algorithm

Our algorithm works in phases, where in each phase we find some edges of T_G^*. The input to Phase i is a planar degree-3 graph G_i. Initially, G is transformed into a degree-3 graph which is the graph G_0. Let T_i^* be the minimum spanning tree of G_i. Phase i proceeds in three steps. First, we find a $(c, 2c^4)$-connector S_i of G_i, where c is a constant (whose value will be determined later). Each tree induced by S_i has size at most $2c^4$ and the non-zero weight edges of S_i belong to T_G^*. Second, we contract each tree induced by S_i in G_i into a single vertex. Let G_i' be the contracted graph. Third, we transform G_i' into a degree-3 graph to meet the input requirements of the next phase. A less informal description of the algorithm follows.

Algorithm: MST-Planar
Input: A planar graph G.
Output: The minimum spanning tree, T_G^*, of G.
1. Transform G into a degree-3 graph G_0;
2. $i = 0$;
3. **While** G_i contains more than one vertex **do**

(a) Find a $(c, 2c^4)$-connector S_i in G_i, where c is a constant;
(b) Contract each tree induced by S_i in G_i and let G'_i be the resulting graph;
(c) Transform G'_i into a degree-3 graph G_{i+1};
(d) $i = i + 1$;

od

4. Return the non-zero weight edges in $\bigcup_{j=0}^{i-1} S_j$, which are the edges of T^*_G;

We first discuss the correctness of the algorithm. Let Z_i denote the set of zero weight edges in G_i.

Lemma 6. $T^*_i = (T^*_{i+1} - Z_{i+1}) \cup S_i = (T^*_{i+1} \cup S_i) - Z_{i+1}$.

Proof. Consider the graph G'_i which is formed by contracting the trees induced by S_i in G_i. By Lemma 2 (iii), we have that $T^*_i = T^*_{G'_i} \cup S_i$. On the other hand, by Lemma 5 we have that $T^*_{G'_i} = T^*_{i+1} - Z_{i+1}$. Therefore, $T^*_i = (T^*_{i+1} - Z_{i+1}) \cup S_i = (T^*_{i+1} \cup S_i) - Z_{i+1}$, where the last equality is true because $S_i \cap Z_j = \emptyset$ for $i \neq j$. □

Lemma 7. *Let t be the total number of iterations of Algorithm MST-Planar. Then, $T^*_G = \bigcup_{i=0}^{t} S_i - \bigcup_{i=0}^{t} Z_i$.*

Proof. By Lemma 5 we have that $T^*_G = T^*_0 - Z_0$. Now, by repeated applications of Lemma 6, we get:

$$\begin{aligned}
T^*_G &= T^*_0 - Z_0 = ((T^*_1 \cup S_0) - Z_1) - Z_0 \\
&= ((((T^*_2 \cup S_1) - Z_2) \cup S_0) - Z_1) - Z_0 \\
&= ((((T^*_2 \cup S_1) \cup S_0) - Z_2) - Z_1) - Z_0 \\
&= \cdots \\
&= ((\cdots((T^*_{t+1} \cup S_t) \cup S_{t-1}) \cdots \cup S_0) - Z_{t+1}) - Z_t) - \cdots) - Z_0 \\
&= \bigcup_{i=0}^{t} S_i - \bigcup_{i=0}^{t} Z_i
\end{aligned}$$

where the last equality follows from the fact that $T^*_{t+1} = Z_{t+1} = \emptyset$ and $\bigcap_{i=0}^{t} Z_i = \emptyset$. □

Hence, the correctness of Algorithm MST-Planar has been established. We now turn to the resource bounds. We shall need the following lemma, whose proof is given in Sect. 4.

Lemma 8. *For all constant c, a $(c, 2c^4)$-connector of a degree-3 graph G with n vertices can be computed: (i) in $O(1)$ time using $O(n)$ work on a CRCW PRAM; (ii) in $O(\log^* n)$ time using $O(n \log^* n)$ work on an EREW PRAM; (iii) in $O(\log n)$ time using $O(n)$ work on an EREW PRAM.*

The transformation of the input graph G into a degree-3 graph G_0, in Step 1 of the algorithm, can be done in $O(\log n)$ time and $O(n)$ work on an EREW PRAM (Lemma 3 (i)). Step 4 can be implemented within the same resource bounds.

Let us now consider the resource bounds of Step 3. Let n_i and n'_i denote the number of vertices in G_i and G'_i, respectively. The bounds of Step 3(a) are given by Lemma 8. In Step 3(b), we have first to contract each tree in G_i induced by S_i. This means that we have to remove internal edges in such a tree and replace multiple edges between two vertices by the one with minimum weight. Since each tree in S_i has size at most $2c^4$, and c is a constant, Step 3(b) takes $O(1)$ time using $O(n_i)$ work on an EREW PRAM. Moreover, since each tree in S_i has size at least c, we have that $n'_i \leq n_i/c$. Note that G'_i is also a (simple) planar graph (Lemma 3 (ii)). Finally, Step 3(c) needs $O(1)$ time using $O(n_i)$ work on an EREW PRAM, by Lemma 3 (i). Consequently, each iteration of the while-loop in Step 3 is dominated by the resource bounds of Step 3(a).

To bound the number of iterations, note that by Lemma 3 (ii), $n_{i+1} \leq 5n'_i - 12 \leq 5n_i/c - 12$. Hence, by choosing $c = 10$, we have that $n_{i+1} \leq n_i/2$. Thus, the number of iterations of the while-loop in Step 3 is $\lceil \log n \rceil$.

Now, on a CRCW PRAM, each iteration takes $O(1)$ time and $O(n_i)$ work, by Lemma 8 (i). Therefore, in total Step 3 can be implemented in $O(\log n)$ time and $O(\sum_{i=0}^{\lceil \log n \rceil} n_i) = O(n)$ work.

Similarly, Step 3 can be implemented in $O(\log n \log^* n)$ time and $O(n \log^* n)$ work on an EREW PRAM, since each iteration takes $O(n_i \log^* n)$ work and $O(\log^* n)$ time by Lemma 8 (ii). To achieve optimal work on an EREW PRAM, we additionally use the implementation of the $(c, 2c^4)$-connector which runs in $O(\log n)$ time and $O(n_i)$ work (Lemma 8 (iii)). Having these two EREW PRAM implementations of a $(c, 2c^4)$-connector, we apply the method given in [19, Sect. 4] or in the proof of Theorem 5.1 in [12]: we run in the first $O(\log^* n)$ iterations the optimal implementation (Lemma 8 (iii)), and in the remaining iterations the non-optimal one (Lemma 8 (ii)). This results (again) in a running time of $O(\log n \log^* n)$, but now a simple simulation argument (see [19, Sect. 4] or [12, Theorem 5.1]) shows that the algorithm can be performed using only $O(n)$ work.

We have therefore established the following.

Theorem 1. *A minimum spanning tree of an n-vertex weighted planar graph G can be found: (i) in $O(\log n)$ time and $O(n)$ work on a CRCW PRAM; (ii) in $O(\log n \log^* n)$ time using $O(n)$ work on an EREW PRAM.*

4 Finding a $(c, f(c))$-Connector in a Degree-3 Graph

In this section, we shall prove Lemma 8. Let H be a degree-3 graph with n vertices. Recall that a $(c, f(c))$-connector of H is a set of edges $S \subseteq T^*_H$ such that for each tree T induced by S in H $c \leq |T| \leq f(c)$, where $|T|$ denotes the number of vertices in T and $f(c)$ is a function of c.

Note that one should be careful in finding a $(c, f(c))$-connector, in the sense that there are many simple ways to do it, but they result in a value for $f(c)$ which may be exponential in c. Hence, a different idea is required in order to avoid such a huge value for $f(c)$. In this section, we shall show how to achieve $f(c) = 2c^4$.

We find the $(c, f(c))$-connector in two stages. In the first stage we find a set of edges $K \subseteq T_H^*$ such that each tree induced by K in H contains at least c vertices. However, there may be some trees having as many as $\Theta(n)$ vertices. Then in the second stage, we remove some edges from K in order to break down these "big" trees into trees of bounded size. The remaining edges in K form a $(c, f(c))$-connector of G.

The first stage consists of a number of iterations, where iteration i finds a set of edges $K_i \subseteq K$. Let F_i be the set of trees induced by K_i in H (where each tree in F_0 is a single vertex). The first stage of the $(c, f(c))$-connector algorithm is implemented as follows.

Stage 1 of the $(c, f(c))$-connector algorithm:
1. $i = 0$; $K_i = \emptyset$; $M = \emptyset$;
2. Let F_i be the set of trees induced by K_i in H;
3. **while** $\exists$ tree T in F_i such that $|T| < c$ **do**
   Find the minimum-weight external edge of T and add it to M;
   $K_{i+1} = K_i \cup M$; Let F_{i+1} be the set of trees induced by K_{i+1} in H;
   $M = \emptyset$; $i = i + 1$;
   od

Observe that each tree in F_i contains at least 2^i vertices. Thus, after $\lceil \log c \rceil$ iterations, every tree contains at least c vertices. Let $K = K_{\lceil \log c \rceil}$. Since only trees with less than c vertices participate in every iteration, it follows that each iteration can be done in $O(\log c)$ time using $O(n)$ work. (*Remark*: within the same resouce bounds we can also check whether $|T| < c$.) Hence, the first stage runs in $O(\log^2 c)$ time using $O(n \log c)$ work.

In the second stage, we have to remove some edges in K to obtain a $(c, f(c))$-connector of H. Let T be a tree induced by K. Recall that every internal vertex in T has degree bounded by 3 and $|T| \geq c$. To find a $(c, f(c))$-connector of H, it suffices to find a $(c, f(c))$-connector in every such tree T; then, the union of the $(c, f(c))$-connectors of every tree T is a $(c, f(c))$-connector of H.

The second stage consists also of a number of iterations. In each iteration we contract a subtree of T (in a manner to be discussed). Let T_i denote T at the beginning of the ith iteration. Initially, $T_0 = T$ and $i = 0$.

Every vertex $v \in T_i$ represents a (contracted) connected subtree of T_{i-1} and hence of T. Let $size(v)$ denote the number of vertices of T that v represents (i.e., they have been contracted into v). Since every vertex of T has degree at most 3, the degree of v in T_i is at most $3size(v) - 2(size(v) - 1) = size(v) + 2$.

Let v be a vertex of T_i. Then: (i) v is called *inactive* if $size(v) \geq c$; (ii) v is called *neutral* if $size(v) < c$ and all adjacent vertices of v are inactive; and (c) v is called *active*, in all other cases. Note that the degree of an active vertex in T_i is at most $c + 1$. Stage 2 is implemented as follows:

Stage 2 of the $(c, f(c))$-connector algorithm:
for every tree T induced by K **pardo**
1. $i = 0$; $T_i = T$; $A_i = \{v | v \in T_i\}$; **forall** $v \in T_i$ **pardo** $size(v) = 1$ **odpar**;

2. **while** $|A_i| > 1$ **do**
   (a) Find a maximal independent set I in A_i, where A_i is the set of active vertices in T_i;
   /* Perform a selective contraction */
   (b) Every $u \in A_i - I$ selects one of its neighbors in I;
   (c) **if** $\exists v \in I$ that has not been selected **then**
   v selects (arbitrarily) one of its neighbors in $A_i - I$;
   (d) The selection process defines connected subtrees of T_i consisting of active vertices. Then, contract each such subtree into a single vertex z and compute $size(z)$;
   (e) Call the resulting tree T_{i+1} and set $i = i + 1$;
   od
3. Every neutral vertex in T_i selects arbitrarily one of its adjacent inactive vertices to hook (i.e., to merge together in a single component). Contract each such component into a single vertex and call the resulting tree T_f;

odpar

At the end of the ith iteration, every active vertex in T_i has *size* at least 2^i. Hence, after at most $\lceil \log c \rceil$ iterations there are no more active vertices. Furthermore, the total number of active vertices that define a connected subtree of T_i (to be contracted) is at most $(c+2)+c(c+1)$ (where the first term comes from Step 2(b) and the second term from Step 2(c)), which is c^2+2c+2. Since every active vertex has *size* at most $c-1$, we have that the *size* of a vertex $u \in T_{i+1}$, at the beginning of the $(i+1)$th iteration, is $size(u) \leq (c^2+2c+2)(c-1) = c^3+c^2-2$.

Let $T_{\lceil \log c \rceil}$ be the tree at the end of Step 2. Note that every vertex in $T_{\lceil \log c \rceil}$ is either inactive or neutral, and that T_f (the tree at the end of Step 3) contains only inactive vertices. Let v be an inactive vertex in $T_{\lceil \log c \rceil}$. Since $size(v) \leq c^3+c^2-2$, there are at most $size(v)+2$ neutral vertices adjacent to it (each of *size* at most $c-1$). Consequently, an inactive vertex $u \in T_f$ has $size(u) \leq c^3+c^2-2+(c^3+c^2)(c-1) = c^4+c^3-2 \leq 2c^4$. Hence, every vertex $u \in T_f$ satisfies $c \leq size(u) \leq 2c^4$ and represents a connected subtree U of T, where $|U| = size(u)$. Moreover, every edge of U belongs to T_H^*, as a consequence of Stage 1. Therefore, the union of all these subtrees U, or alternatively the set of edges in $T - T_f$, constitutes the required $(c, f(c))$-connector of T, where $f(c) = 2c^4$.

Let us now consider the complexity of Stage 2. Observe that since T is a tree, no multiple edges are created in every contraction of a subtree of T, and consequently there is no need to invoke a sorting procedure to eliminate all but one multiple edges. Hence, every contraction of a subtree of T of size p can be done in $O(\log p)$ time and $O(p)$ work on an EREW PRAM.

We first discuss the EREW PRAM complexity. Step 3 takes $O(\log c)$ time and $O(n \log c)$ work over all trees T. The resource bounds of each iteration of Step 2 are dominated by those of Step 2(a) to find a maximal independent set in a tree. This requires (over all trees T) $O(\log^* n)$ time and $O(n \log^* n)$ work [16, Theorem 4], or alternatively $O(\log n)$ time and $O(n)$ work [17, Lemma 7]. Hence, Stage 2 runs in $O(\log^2 c \log^* n)$ time and $O(n \log c \log^* n)$ work, or in $O(\log^2 c \log n)$ time and $O(n \log c)$ work.

We now turn to the CRCW PRAM complexity. Unfortunately, the maximal independent set algorithm of [16] cannot take advantage of this model to solve the problem faster. Instead, we use an algorithm from [3] that computes a so-called *fractional* independent set. More precisely, the following result is proved in [3, Sect. 6]: Let G be an n-vertex graph of constant degree. Then, an independent set I such that $|I| \geq \varepsilon n$, where $0 < \varepsilon < 1$ is a constant, can be found in $O(1)$ time and $O(n)$ work on a CRCW PRAM.

Now, in Step 2(a) we do not find a maximal independent set, but a fractional one using the algorithm of [3]. Note that in this case we cannot guarantee that at the end of the ith iteration every active vertex has *size* at least 2^i. However, we can guarantee that in every iteration a constant fraction of the (remaining) active vertices $\varepsilon|A_i|$ perform a selective contraction. Consequently, at the beginning of the next iteration, there are at most $(1-\varepsilon)|A_i| \leq (1-\varepsilon)|T_i| \leq (1-\varepsilon)^i|T_0|$ active vertices. Hence, we need $\lceil \log_{(\frac{1}{1-\varepsilon})} c \rceil = O(\log c)$ iterations to eliminate all active vertices. This implies that Stage 2 can be implemented to run in $O(\log^2 c)$ time and $O(n \log c)$ work on a CRCW PRAM.

The bounds of Lemma 8 follow now from the fact that c is always a constant in all applications of the $(c, f(c))$-connector algorithm with $f(c) = 2c^4$.

5 Extensions of Our Results

Following [17], a class $\mathcal{G}$ of undirected graphs is called *linearly contractible* if: (1) for all G in $\mathcal{G}$ $m \leq kn$, where k is a constant; and (2) $\mathcal{G}$ is closed under taking of minors, i.e., every subgraph and every *elementary contraction* of a graph in $\mathcal{G}$ is in $\mathcal{G}$. An *elementary contraction* of a graph G is a new graph obtained from G by contracting two adjacent vertices u and v into a single vertex z. Examples of the class of linearly contractible graphs are planar graphs, graphs of bounded treewidth and graphs of bounded genus.

Observe that the only properties that our algorithm requires from a planar graph are those stated in Lemma 2 (i) and (ii), and which are included in the above definition of the linearly contractible graphs. Hence, Lemma 2 is satisfied (in a sense) by any graph belonging to a linearly contractible class, with the difference that part (i) now becomes $m \leq kn$. This implies that the number n_h of vertices of the transformed graph (Lemma 3) is now bounded by $n_h \leq 2m - n = (2k-1)n$. To achieve again a number of $\lceil \log n \rceil$ iterations of our MST algorithm in Sect. 3, it suffices to choose $c = 4k - 2$ in the construction of the $(c, 2c^4)$-connector. We have therefore established the following.

Theorem 2. *A minimum spanning tree of an n-vertex weighted graph G, drawn from a linearly contractible class, can be found: (i) in $O(\log n)$ time and $O(n)$ work on a CRCW PRAM; (ii) in $O(\log n \log^* n)$ time using $O(n)$ work on an EREW PRAM.*

We can further achieve optimal results in the case of graphs with non-bounded genus. The idea is as follows.

Let G be a graph with genus γ. Then, $m \leq 3n + 6\gamma - 6$ [20]. Note that when G is contracted, the genus of the resulting graph may remain unchanged and the total number of edges may not decrease accordingly. However, if γ is very small compared to the number of vertices in the graph, the total number of edges contributed by the "6γ" term is also very small. In particular, we can assume that $m \leq 4n$, when $\gamma = o(n)$. Therefore, our algorithm can still work properly for a number of iterations, as long as the condition $\gamma = o(n_i)$ in every iteration is satisfied (n_i being the number of vertices at iteration i), for some suitable value of γ and choice of c. As soon as, after a particular iteration, $\gamma \geq n_i$, we switch to another algorithm. Next we show that if $\gamma \leq 2^{\frac{\log n}{\log\log n}}$, the above approach gives an optimal algorithm to compute T^*_G. Note that $2^{\frac{\log n}{\log\log n}} = \Omega(\text{poly}\log n)$.

We find T^*_G in two phases. In Phase I, we run Algorithm MST-Planar up to the ith iteration, where $i = \log n - \log n/\log\log n$ and we choose the constant c to be 14. At the end of Phase I, we obtain a graph G_{i+1} in which the total number of vertices is no more than $2^{\frac{\log n}{\log\log n}}$. In Phase II, we use the algorithm of [8] to find T^*_{i+1} in G_{i+1}. The edges found in the two phases form the MST of G.

Phase I takes $O(\log n \log^* n)$ time using $O(n)$ work on an EREW PRAM, or $O(\log n)$ time using $O(n)$ work on a CRCW PRAM.

Phase II takes $O(\log n' \log\log n')$ time using $O(m' \log n' \log\log n')$ work. As $n' = 2^{\frac{\log n}{\log\log n}}$, Phase II runs in $O(\log n)$ time using no more than $O(n)$ work on an EREW PRAM. Thus we have established the following.

Theorem 3. *A minimum spanning tree of an n-vertex weighted graph G with genus $\gamma \leq 2^{\frac{\log n}{\log\log n}}$ can be found: (i) in $O(\log n)$ time and $O(n)$ work on a CRCW PRAM; (ii) in $O(\log n \log^* n)$ time using $O(n)$ work on an EREW PRAM.*

6 Conclusions

We presented an $O(n)$ work parallel algorithm for solving the MST problem on planar, minor closed, and a class of non-bounded genus graphs. The algorithms runs in $O(\log n \log^* n)$ time on an EREW PRAM and in $O(\log n)$ time on a CRCW PRAM.

An interesting open problem is to develop a $O(n)$-work deterministic EREW PRAM algorithm for these graph classes that runs in $O(\log n)$ time.

Acknowledgements. The last author is indebted to his mentor Paul Spirakis, who taught him by example to be a scientist and who uniquely affected the shaping of his career.

References

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows. Prentice-Hall, Englewood (1993)
2. Awerbuch, B., Shiloach, Y.: New connectivity and MSF algorithms for shuffle-exchange network and PRAM. IEEE Trans. Comput. **36**(10), 1258–1263 (1987)

3. Bodlaender, H., Hagerup, T.: Parallel algorithms with optimal speedup for bounded treewidth. SIAM J. Comput. **27**(6), 1725–1746 (1998)
4. Boruvka, O.: O jistém problému minimálním. Práca Moravské Přírodovědecké Společnosti **3**, 37–58 (1926)
5. Chazelle, B.: A minimum spanning tree algorithm with inverse-Ackermann type complexity. J. ACM **47**(6), 1028–1047 (2000)
6. Cheriton, D., Tarjan, R.E.: Finding minimum spanning trees. SIAM J. Comput. **5**, 724–742 (1976)
7. Chin, F.Y., Lam, J., Chen, I.N.: Efficient parallel algorithms for some graph problems. Commun. ACM **25**(9), 659–665 (1982)
8. Chong, K.W.: Finding minimum spanning trees on the EREW PRAM. In: Proceedings of the International Computer Symposium—ICS'96, pp. 7–14. Taiwan (1996)
9. Chong, K.W., Lam, T.W.: Finding connected components in $O(\log n \log\log n)$ time on the EREW PRAM. J. Algorithms **18**, 378–402 (1995)
10. Chong, K.W., He, Y., Lam, T.W.: Concurrent threads and optimal parallel minimum spanning trees algorithm. J. ACM **48**(2), 297–323 (2001)
11. Cole, R., Klein, P.N., Tarjan, R.E.: Finding minimum spanning forests in logarithmic time and linear work using random sampling. In: Proceedings of the 8th ACM symposium on Parallel Algorithms and Architectures (ACM), pp. 243–250 (1996)
12. Cole, R., Vishkin, U.: Deterministic coin tossing with applications to optimal parallel list ranking. Inf. Control **70**, 32–53 (1986)
13. Cole, R., Vishkin, U.: Approximate and exact parallel scheduling with applications to list, tree and graph problems. In: Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, pp. 478–491. IEEE (1986)
14. Fredman, M., Willard, D.E.: Trans-dichotomous algorithms for minimum spanning trees and shortest paths. J. Comput. Syst. Sci. **48**, 533–551 (1994)
15. Gabow, H., Galil, Z., Spencer, T., Tarjan, R.E.: Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica **6**(2), 109–122 (1986)
16. Goldberg, A., Plotkin, S., Shannon, G.: Parallel symmetry-breaking in sparse graphs. SIAM J. Discrete Math. **1**, 434–446 (1988)
17. Hagerup, T.: Optimal parallel algorithms on planar graphs. Inf. Comput. **84**, 71–96 (1990)
18. Hagerup, T.: Optimal Parallel Computation of Minimum Spanning Forests in Planar Graphs, Technical Report 11/1990. Universität des Saarlandes, May 1990
19. Hagerup, T., Chrobak, M., Diks, K.: Optimal parallel 5-colouring of planar graphs. SIAM J. Comput. **18**(2), 288–300 (1989)
20. Harary, F.: Graph Theory. Addison-Wesley, Reading (1969)
21. JáJá, J.: An Introduction to Parallel Algorithms. Addison-Wesley, Reding (1992)
22. Johnson, D.B., Metaxas, P.: Connected components in $O(\log^{3/2}|V|)$ parallel time for the CREW PRAM. In: Proceeings of 32nd IEEE Symposium on Foundations of Computer Science, pp. 688–695, IEEE (1991)
23. Johnson, D.B., Metaxas, P.: A parallel algorithm for computing minimum spanning trees. J. Algorithms **19**, 383–401 (1995)
24. Karger, D.R.: Approximating, verifying, and constructing minimum spanning trees. Unpublished manuscript (1992)
25. Karger, D.R., Klein, P.N., Tarjan, R.E.: A randomized linear-time algorithm to find minimum spanning trees. J. ACM **42**(2), 321–328 (1995)

26. Karp, R., Ramachandran, V.: Parallel Algorithms for Shared-Memory Machines. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. A, pp. 869–941. Elsevier, Amsterdam (1990)
27. Pettie, S., Ramachandran, V.: An optimal minimum spanning tree algorithm. J. ACM **49**(1), 16–34 (2002)
28. Pettie, S., Ramachandran, V.: A randomized time-work optimal parallel algorithm for finding a minimum spanning forest. SIAM J. Comput. **31**(6), 1879–1895 (2002)
29. Zaroliagis, C.D.: Simple and work-efficient parallel algorithms for the minimum spanning tree problem. Parallel Process. Lett. **7**(1), 25–37 (1997)