Competitive Maintenance of Minimum Spanning Trees in Dynamic Graphs*

Extended abstract

Miroslaw Dynia[1], Miroslaw Korzeniowski[2**], and Jarosław Kutyłowski[3]

[1] DFG Graduate College "Automatic Configuration in Open Systems", Heinz Nixdorf Institute, University of Paderborn, Germany
[2] Institute of Computer Science, University of Wroclaw, Poland and LaBRI – INRIA Futurs, Bordeaux, France
[3] International Graduate School of Dynamic Intelligent Systems, Heinz Nixdorf Institute, University of Paderborn, Germany

Abstract. We consider the problem of maintaining a minimum spanning tree within a graph with dynamically changing edge weights. An online algorithm is confronted with an input sequence of edge weight changes and has to choose a minimum spanning tree after each such change in the graph. The task of the algorithm is to perform as few changes in its minimum spanning tree as possible.

We compare the number of changes in the minimum spanning tree produced by an online algorithm and that produced by an optimal offline algorithm. The number of changes is counted in the number of edges changed between spanning trees in consecutive rounds.

For any graph with n vertices we provide a deterministic algorithm achieving a competitive ratio of $\mathcal{O}(n^2)$. We show that this result is optimal up to a constant. Furthermore we give a lower bound for randomized algorithms of $\Omega(\log n)$. We show a randomized algorithm achieving a competitive ratio of $\mathcal{O}(n \log n)$ for general graphs and $\mathcal{O}(\log n)$ for planar graphs.

1 Introduction

We consider the problem of maintaining a minimum spanning tree by an online algorithm with an adversary changing weights of edges of the underlying graph.

* Partially supported by the EU within the 6th Framework Programme under contract 001907 (DELIS) and by the DFG-Sonderforschungsbereich SPP 1183: "Organic Computing. Smart Teams: Local, Distributed Strategies for Self-Organizing Robotic Exploration Teams".

** This work was done while the author was in the International Graduate School of Dynamic Intelligent Systems, Heinz Nixdorf Institute, University of Paderborn, Germany. The author is partially supported by MNiSW grant number N206 001 31/0436, 2006-2008.

Every time the weight of an edge is changed, the algorithm must output a minimum spanning tree for the new graph. If possible, this spanning tree should be the same spanning tree as computed in the previous round or at least both trees should be similar. For every edge changed in the minimum spanning tree between consecutive rounds, the algorithm is charged unit cost.

The problem of mainting a minimum spanning tree after the underlying graph is changed has been widely studied in literature (see e.g. [9, 10, 1, 5, 8]). Typically only the computational effort needed to maintain the minimum spanning tree, i.e. to choose a proper minimum spanning tree after an edge weight has been changed, has been considered. This research has resulted in the development of very efficient data structures which maintain information about the graph and allow to calculate new minimum spanning trees fast. On the other hand, in many applications the computational complexity needed for computing a new minimum spanning tree is not the only important factor. Another complexity parameter is the number of changes in the minimum spanning tree between rounds. Here, the main problem lies in choosing a minimum spanning tree from the set of possible minimum spanning trees. The chosen MST should retain its minimality property for a long time.

In our model changing the minimum spanning tree by an algorithm is considered the most costly factor. The question on how to compute new minimum spanning trees has been already well studied, so that we do not consider it in our paper. We want to motivate this setting by giving an example coming from our research in the area of mobile networks. We also show further application areas of the results presented in this paper.

We consider mobile networks which are wide-spread over large terrain. The network as a whole has its tasks to perform in the terrain and needs a communication structure so that individual participants can communicate with each other and coordinate. Due to the large size of the terrain and heavy environmental conditions, the transmission power of the network participants may not suffice to form a connected multihop network. This may happen for example in a mountainous area during rescue actions, when satellite communication is not available and mountains strongly hinder radio wave propagation. To ensure a communication framework connecting network parts, we propose to use mobile robots, serving as relay stations which form multihop communication paths between network parts. These small robots have no other function as to keep their position on the communication path and to forward messages along the established relay path. This is a new approach, and the authors are not aware of a similar solution presented in the literature.

We can model network parts as nodes of a graph and paths in the terrain between them as edges of this graph. Obviously, the paths have different lengths and these are mapped to edge weights of the graph. Our goal is to create a communication structure between network parts by using mobile relay stations on some of the paths. These relay stations must be located on the path in some limited distance to be able to communicate with each other – consequently the number of required relay stations per path is proportional to its length. We

want to select the path to be populated by relay stations so that the network is connected and simultaneously to minimize the number of used relay stations. Minimum spanning trees in this graph are the optimal solutions regarding this measure. The minimum spanning tree must be maintained while the graph dynamically changes – the weights of edges increase or decrease while nodes move. The goal of an algorithm should be not only to maintain a minimum spanning tree all the time but also to minimize the number of changes in the spanning tree. The rationale for this is that every change in the spanning tree forces mobile relay stations to move from one path to another, incurring large energy and time cost. This cost is surely larger than the cost of computing a new minimum spanning tree and thus the primary goal should be to minimize the number of changes in the minimum spanning tree.

Apart from the scenario from mobile network research, the described cost model is reasonable in any application where changes in the minimum spanning tree are costly. This occurs e.g. in networks where trees are used as a routing or overlay structure and changing the MST means that routing or configuration tables have to be broadcasted along the network and updated. Minimum spanning trees have been used in such scenarios for a long time, some recent examples may be found in [15, 16, 12].

Our model does not explicitely require that the graph contains many different minimum spanning trees, but the application of our algorithms is only reasonable when this occurs. In such graphs, both bad and good choices for the minimum spanning tree can be made. Graphs contain many minimum spanning trees when there are many edges with equal weight. This is quite improbable to happen if the edge weights are given by sensor readings, as described in the previous paragraph. On the other hand, small fluctuations of these sensor readings can cause the only minimum spanning tree to change very frequently. Thus, we recommend to round the sensor readings appropriately, so that some stabiblity is brought into the edge weights (without sacrificing accuracy) and fluctuations can be reduced. Then, presented competitive algorithms can show their power when it comes to choosing the proper minimum spanning tree from those available at a moment.

1.1 Our contribution

We compare the performance of our algorithms regarding the described cost model to the performance of an optimal offline algorithm by competitive analysis. We present a detailed model for the mentioned problem and give lower and upper bounds for deterministic online algorithms (see Sections 2 and 3). Our deterministic algorithm achieves a competitive ratio of $n^2/2$ and this is optimal up to a constant factor. We improve the competitive ratio by introducing a randomized algorithm with an expected competitive ratio of $\mathcal{O}(n \log n)$ and $\mathcal{O}(\log n)$ for planar graphs. The discussion of the planar case can be found in the full version of this paper, together with a lower bound of $\Omega(\log n)$ for the expected competitive ratio of any randomized algorithm.

The mentioned randomized algorithm works only for a restricted scenario, where the weights of edges can only grow. In this context, it is worth noting that

the lower bound presented in Section 2 does not need to decrease edge weights. This gives some indication that the real hardness of the problem does not lie within decreasing edges, but can be also expressed by only increasing the weights of edges.

1.2 Related work

Research on minimum spanning trees dates back to textbook algorithms by Kruskal [11] and Prim [14]. In the static setting improved solutions have been considered e.g in [4]. All this work assumes that the graph remains static and considers the classical runtime complexity of algorithms. Research in this area is still vivid, see e.g. recent results by Chazelle [3] and Pettie [13].

As already noted, large effort has been put into constructing data structures which also allow minimum spanning trees to be computed efficiently when changes in the structure of the graph occur. These changes can either concern edge weights as assumed in our work (see e.g. [9]) or might encompass adding and deleting vertices ([5, 8, 10]). Furthermore kinetic spanning trees have been considered in [1] to model the changes of edge lengths in a more predictable way.

For an in-depth survey of different types of subproblems in the area of minimum spanning trees, for applications and results we refer the interested reader to [7].

1.3 Our model

We model a mobile ad-hoc network as a connected graph $G = (V, E)$ with edges weighted initially by the function $w^0 : E \to \mathbb{N}_+$. Time is divided into discrete time steps called rounds. In round i the value of $\sigma(i) \in E \times \{-1, +1\}$ defines the change of weight of an edge. Only one edge changes its weight in one round and the change is bounded to either $+1$ or -1. The sequence σ is the input sequence.

Basing on the original input sequence we denote for convenience of notation by $\delta(i, e) \in \{-1, 0, 1\}$ the change of weight of edge $e \in E$ in the i-th round. Formally we have

$$\delta(i,e) = \begin{cases} -1 \ , & \text{if } \sigma(i) = (e,-1) \\ 1 \quad , & \text{if } \sigma(i) = (e,+1) \\ 0 \quad , & \text{otherwise .} \end{cases}$$

Furthermore, we introduce the function $w : \mathbb{N}_+ \times E \to \mathbb{N}_+$ which maps a round number r and edge e to the edge weight at the beginning of round r. This gives

$$w^r(e) = w^0(e) + \sum_{i=1}^{r-1} \delta(i,e) \ .$$

An algorithm ALG solving the Online Dynamic Minimum Spanning Tree (ODMST) problem reads the input sequence σ and after obtaining $\sigma(r)$ outputs a minimum spanning tree, denoted by $\mathcal{M}^r_{\text{ALG}}$. Since the algorithm does not know the future, it has no access to the values of $\sigma(i)$ for $i > r$ in round r. The cost of

an algorithm in round r is defined as the number of edges in which $\mathcal{M}^{r-1}_{\mathrm{ALG}}$ and $\mathcal{M}^{r}_{\mathrm{ALG}}$ differ, formally

$$C^r_{\mathrm{ALG}} := \left|\{e \in E | e \notin \mathcal{M}^{r-1}_{\mathrm{ALG}} \wedge e \in \mathcal{M}^{r}_{\mathrm{ALG}}\}\right| \ .$$

Additionally $C_{\mathrm{ALG}}(\sigma)$ is the cost of the algorithm on the whole sequence, thus

$$C_{\mathrm{ALG}}(\sigma) = \sum_{i=1}^{|\sigma|} C^i_{\mathrm{ALG}} \ .$$

To measure the performance of algorithms solving the ODMST problem we employ competitive analysis (see e.g. [2]). The definition of the ODMST problem fulfills the typical definition of online problems. An optimal algorithm OPT computes a sequence of minimum spanning trees $\mathcal{M}^i_{\mathrm{OPT}}$ for $i = 1, \ldots, |\sigma|$ minimizing the value of $C_{\mathrm{OPT}}(\sigma)$, with $C_{\mathrm{OPT}}(\sigma) = \sum_{i=1}^{|\sigma|} C^i_{\mathrm{OPT}}$, where

$$C^r_{\mathrm{OPT}} := \left|\{e \in E | e \notin \mathcal{M}^{r-1}_{\mathrm{OPT}} \wedge e \in \mathcal{M}^{r}_{\mathrm{OPT}}\}\right| \ .$$

The optimal algorithm has access to the whole sequence σ in advance.

A deterministic algorithm ALG has a competitive ratio of $\mathcal{R}_{\mathrm{ALG}}$ if for all input sequences σ we have $C_{\mathrm{ALG}}(\sigma) \leq \mathcal{R}_{\mathrm{ALG}} \cdot C_{\mathrm{OPT}}(\sigma) + c$, where c does not depend on σ.

For a randomized algorithm ALG we have to introduce the notion of an oblivious adversary. The input sequence σ is constructed by an adversary having access to the algorithm ALG and the probability distributions used by ALG to perform its task. The oblivious adversary does not know the random bits used by ALG. With the given notion a randomized algorithm ALG has a competitive ratio of $\mathcal{R}_{\mathrm{ALG}}$ if $\mathrm{E}[C_{\mathrm{ALG}}(\sigma)] \leq \mathcal{R}_{\mathrm{ALG}} \cdot C_{\mathrm{OPT}}(\sigma) + c$ for every input sequence σ. The expected value of C_{ALG} is taken with respect to the random choices of ALG.

1.4 Notation

The following notation will be used throughout the whole paper. The set of alternative edges $\mathcal{A}(e, r)$ is defined for a graph $G = (V, E)$, a round r, an algorithm ALG and an edge $e \in \mathcal{M}^r_{\mathrm{ALG}}$. Removing e from $\mathcal{M}^r_{\mathrm{ALG}}$ splits the tree into two parts. Consider the vertex sets V_1 and V_2 of both parts. Then the set of edges on the cut between V_1 and V_2 is denoted by

$$\mathcal{A}(e, r) = \{(u, v) \in E | u \in V_1 \wedge v \in V_2\} \ .$$

We also define a set of alternatives which have a certain weight

$$\mathcal{A}_{\mathrm{W}}(e, r, w) = \{e' \in \mathcal{A}(e, \mathcal{M}^r_{\mathrm{ALG}}) | w^r(e') = w\} \ .$$

Suppose we extend $\mathcal{M}^r_{\mathrm{ALG}}$ by adding an edge e and thus creating a cycle in $\mathcal{M}^r_{\mathrm{ALG}}$. Then all edges on this cycle except for e are denoted by $\mathcal{C}(e, r)$. Analogously to the set $\mathcal{A}_{\mathrm{W}}(\cdot)$, we define a set of all edges from $\mathcal{C}(e, r)$ with a certain weight

$$\mathcal{C}_{\mathrm{W}}(e, r, w) = \{e' \in \mathcal{C}(e, r) | w^r(e') = w\} \ .$$

Note that $\mathcal{A}(e, r)$ includes e, whereas $\mathcal{C}(e, r)$ does not.

2 Deterministic lower bound

We construct an input sequence for the ODMST problem which causes every online deterministic algorithm to have a competitive ratio $\mathcal{R}_{\text{ALG}} \in \Omega(n^2)$. We assume that the input sequence is given by an adversary who examines the moves of ALG. To construct a deterministic lower bound we have to be able to construct an input sequence σ such that for an arbitrary large k we have $C_{\text{OPT}}(\sigma) \geq k$ and $C_{\text{ALG}}(\sigma) \leq \mathcal{R}_{\text{ALG}} \cdot C_{\text{OPT}}(\sigma) + c$ with an appropriate constant c only dependent on the input graph for any deterministic algorithm ALG. This is analogous to the formulation found in [6] given for randomized algorithms, rewritten for deterministic algorithms here.

Input graph. We first describe the graph for the lower bound construction. We take a complete graph $G = (V, E)$ with $|V|$ even and partition V into two sets V_1 and V_2 with $|V_1| = |V_2|$. Call E_C the set of edges lying on the cut between V_1 and V_2. To each edge $e \in E_C$ we assign a weight $w^0(e) = n = |V|$, all other edges are assigned a weight of 1. Obviously at least one edge from E_C must be used in every spanning tree and, since we consider minimum spanning trees, it will be the only one.

Input sequence. We construct an input sequence consisting of phases of length $2|E_C|$. Within each phase ALG has a cost of at least $|E_C|$ and OPT has a cost of 1 or 2. For each k we can concatenate k phases and obtain an input sequence σ for which $k \leq C_{\text{OPT}}(\sigma) \leq 2k$ and $C_{\text{ALG}}(\sigma) \geq \frac{|E_C-1|}{2} C_{\text{OPT}}(\sigma)$. From this fact we can conclude that every deterministic algorithm has a competitive ratio greater or equal $\frac{|E_C-1|}{2}$.

In the first part of a phase, the adversary's goal is to increase the weight of all edges in E_C to $n+1$. The adversary watches the moves of ALG and always increases the weight of an edge from E_C used by ALG. Obviously, ALG can only use edges from E_C with weight n while such edges exist – if it was using one with weight $n+1$, its spanning tree would not be minimal. Thus the adversary is able to increase the weight of an edge used by ALG until all edges have weight $n+1$. Every such change except the last one incurs at least a cost of 1 to the algorithm. Since there are $|E_C|$ requests, the algorithm has a cost of at least $|E_C| - 1$. In the second part of a phase the weight of all edges is decreased to n in the same order as they were increased. We neglect ALG's cost during these operations.

For such an input sequence it is easy to construct a strategy for OPT which has a cost of 1 or 2 in each phase. Then we can construct an input sequence σ such that $C_{\text{OPT}}(\sigma) \geq k$ for every k and $C_{\text{ALG}}(\sigma) \geq \frac{k(|E_C|-1)}{2}$. It follows that the competitive ratio is at least $\frac{|E_C|-1}{2}$ for every phase. Concluding, we have shown that for every online deterministic algorithm ALG for the ODMST problem we have $\mathcal{R}_{\text{ALG}} \in \Omega(n^2)$.

3 Deterministic algorithm MSTMark

In this section we present the deterministic algorithm MSTMark which achieves an optimal, up to a constant factor, competitive ratio for the ODMST problem.

Notation. The MSTMark algorithm (Algorithm 1) works on a graph $G = (V, E)$ computing a minimum spanning tree $\mathcal{M}^r_{\mathrm{ALG}}$ in each round r. Where clear from the context we will write $\mathcal{M}_{\mathrm{ALG}}$ instead of $\mathcal{M}^r_{\mathrm{ALG}}$ omitting the current round number. The minimum spanning tree maintained by the optimal offline algorithm is described as $\mathcal{M}^r_{\mathrm{OPT}}$ and, wherever possible by $\mathcal{M}_{\mathrm{OPT}}$. We say that an algorithm substitutes edge e with e' in round r if we have $\mathcal{M}^{r+1}_{\mathrm{ALG}} = (\mathcal{M}^r_{\mathrm{ALG}} \setminus \{e\}) \cup \{e'\}$.

MSTMark *algorithm.* The algorithm has to respond to two different kinds of events – increases and decreases of weights of edges in G. If the weight of an edge $e \in \mathcal{M}^{r-1}_{\mathrm{ALG}}$ is increased in round r, MSTMark tries to find a suitable alternative $e' \in \mathcal{A}_{\mathrm{W}}(e, r-1, w^{r-1}(e))$. If a not marked edge e' can be found, MSTMark replaces e with e' in $\mathcal{M}^r_{\mathrm{ALG}}$. By the construction of the set $\mathcal{A}_{\mathrm{W}}(\cdot)$ any such edge causes $\mathcal{M}_{\mathrm{ALG}}$ to remain a minimum spanning tree. If an appropriate edge cannot be found, MSTMark sets $\mathcal{M}^r_{\mathrm{ALG}} = \mathcal{M}^{r-1}_{\mathrm{ALG}}$.

If the weight of an edge $e \notin \mathcal{M}^{r-1}_{\mathrm{ALG}}$ is decreased in round r, MSTMark checks whether there is a not marked edge $e' \in \mathcal{C}(e, r-1)$ with a higher weight than $w^r(e)$. If yes, it substitutes e' with e within $\mathcal{M}_{\mathrm{ALG}}$. If no, MSTMark sets $\mathcal{M}^r_{\mathrm{ALG}} = \mathcal{M}^{r-1}_{\mathrm{ALG}}$.

In all other cases MSTMark does not perform any changes in its minimum spanning tree.

The greedy approach changing only one edge of the MST on updates of edge weight has been already successfully applied in algorithms for updating minimum spanning trees, e.g. in [9], thus we won't argue its correctness. Mentioned results also allow to perform the described changes in the minimum spanning tree efficiently by employing appropriate data structures.

Marking with flags. In addition to the described behavior, MSTMark marks edges of G with two kinds of flags: PRESENCE and ABSENCE. The idea is that a flag is put on an edge e, where MSTMark is sure that, respectively, $e \in \mathcal{M}_{\mathrm{OPT}}$ or $e \notin \mathcal{M}_{\mathrm{OPT}}$. This information is only guaranteed for the very round when the mark has been set – for future rounds it may not hold anymore.

For the analysis of the competitive ratio of MSTMark one has to introduce the notion of epochs. The PRESENCE and ABSENCE flags are the key to this analysis. An epoch starts when all flags are removed from the graph (at lines 6, 11, 20 or 25 of MSTMark) and lasts until the next removal of all flags. We can show that in each epoch OPT performs at least one change in its minimum spanning tree and that MSTMark performs at most $n^2/2$ changes. Then, the competetive ratio $\mathcal{R}_{\mathrm{MSTMark}} \leq n^2/2$. The analysis together with technical lemmas can be found in the full version of the paper.

Algorithm 1 MSTMARK(round r)

if weight of edge $e \in \mathcal{M}_{\mathrm{ALG}}^{r-1}$ increased **and** $\mathcal{A}_{\mathrm{W}}(e, r-1, w^{r-1}(e)) \neq \emptyset$ **then**
  $\mathcal{A}_{NM} \leftarrow \{e' \in \mathcal{A}_{\mathrm{W}}(e, r-1, w^{r-1}(e)) | e'$ isn't marked with ABSENCE$\}$
  if $\mathcal{A}_{NM} \neq \emptyset$ **then**
    remove e from $\mathcal{M}_{\mathrm{ALG}}^{r}$ and substitute it with any $e' \in \mathcal{A}_{NM}$
    if e is marked with PRESENCE **then**
      remove all marks
    end if
    mark e with ABSENCE
  else
    remove e from $\mathcal{M}_{\mathrm{ALG}}^{r}$ and substitute it with $e' \in \mathcal{A}_{\mathrm{W}}(e, r-1, w^{r-1}(e))$
    remove all marks
    mark e with ABSENCE
  end if
end if
if weight of edge $e \notin \mathcal{M}_{\mathrm{ALG}}^{r-1}$ decreased **and** $\mathcal{C}_{\mathrm{W}}(e, r-1, w^{r-1}(e)) \neq \emptyset$ **then**
  $\mathcal{C}_{NM} \leftarrow \{e' \in \mathcal{C}_{\mathrm{W}}(e, r-1, w^{r-1}(e)) | e'$ isn't marked with PRESENCE$\}$
  if $\mathcal{C}_{NM} \neq \emptyset$ **then**
    remove e from $\mathcal{M}_{\mathrm{ALG}}^{r}$ and substitute it with any $e' \in \mathcal{C}_{NM}$
    if e is marked with ABSENCE **then**
      remove all marks
    end if
    mark e' with PRESENCE
  else
    remove e from $\mathcal{M}_{\mathrm{ALG}}^{r}$ and substitute it with $e' \in \mathcal{A}_{\mathrm{W}}(e, r-1, w^{r-1}(e))$
    remove all marks
    mark e' with PRESENCE
  end if
end if

4 Randomized algorithm RandMST

The randomized algorithm RANDMST presented in this section achieves an expected competitive ratio of $\mathcal{O}(n \log n)$. This algorithm works only for a limited scenario, where weights of edges are only increased. It is a cut down version of the MSTMARK algorithm, with the handling of flags removed. In the considered scenario edge weights can only grow and we will see that flags are not necessary any more.

Consider a round r in which the weight of an edge $e \in \mathcal{M}_{\mathrm{ALG}}$ is increased. If there exist alternative edges for e with weight $w^{r-1}(e)$, then RANDMST selects one of these edges uniformly at random and uses it instead of e in $\mathcal{M}_{\mathrm{ALG}}^{r}$. In other cases RANDMST ignores the edge weight increase.

We can show that the RANDMST algorithm has an expected competitive ratio of $\mathcal{O}(n \log n)$ for general graphs. For planar graphs the ratio drops to $\mathcal{O}(\log n)$. This improvement is mainly (but not fully) due to the smaller number of edges in a planar graph, comparing to a general graph. The analysis of the planar case can be found in the full version of the paper.

Algorithm 2 RANDMST

1: **if** weight of edge $e \in \mathcal{M}_{\text{ALG}}^{r-1}$ increased in round r **and** $\mathcal{A}_{\text{W}}(e, r-1, w^{r-1}(e)) \neq \emptyset$ **then**
2: $e' \leftarrow$ choose uniformly at random an edge out of $\mathcal{A}_{\text{W}}(e, r-1, w^{r-1}(e))$
3: remove e from $\mathcal{M}_{\text{ALG}}^{r}$ and substitute it with e'
4: **end if**

The Analysis. The idea of the analysis is to consider the behavior of RANDMST in layers separately. Intuitively, a layer w consists only of edges which have weight w. In every layer we will divide the graph G into parts, called fixed components. The idea of fixed components is that the cost of OPT on the whole input sequence is at least as high as the number of fixed components created in all layers. We will also be able to bound the expected cost of RANDMST to $\mathcal{O}(n \log n)$ times the number of fixed components created. From this we will be able to conclude that the expected competitive ratio of RANDMST is at most $\mathcal{O}(n \log n)$.

Certain lemmas from this section have technically involved proofs – these can be found in the full version of this paper. We start the analysis with introducing the notions of fixed components, edge sets and realms.

The fixed components. As already mentioned, we consider a separate layer of fixed components for each weight w. Let $V(G')$ denote the set of vertices of graph G', and $E(G')$ the set of edges of G'. A fixed component is a subgraph of G and in every layer there is exactly one fixed component prior to round 1. A fixed component F in layer w splits if the weight of an edge $e = (u, v) \in E(F)$ with $w(e) = w$ is increased, $e \in \mathcal{M}_{\text{ALG}}$, and the size of minimum spanning trees does not increase (i.e. RANDMST must perform a change in $\mathcal{M}_{\text{ALG}}$). Then the fixed component F splits into two fixed components F_1 and F_2, such that $V(F_1) \subset V(F)$ and $V(F_2) \subset V(F)$ and $V(F_1) \cup V(F_2) = V(F)$. Furthermore, a vertex $x \in V(F)$ is in $V(F1)$ if it can be reached from u when using only edges from $\mathcal{M}_{\text{ALG}} \setminus \{e\}$. Analogously the fixed component F_2 consists of vertices which can be reached from v. We say that a fixed component splits on edge e if the split occurs due to an increase of weight of e. Note, that fixed components form a partition of the vertex set of G, and thus there are at most n fixed components in any layer. It is necessary for the splitting technique to work properly that the part of $\mathcal{M}_{\text{ALG}}$ contained in a fixed component is connected. This property will be established by Lemma 2.

Besides fixed components, we also have edge sets in each layer w. Before round 1 there are no edge sets in any layer. If a fixed component F splits into F_1 and F_2 on an edge e with weight w, an edge set between F_1 and F_2 is created, denoted by $E_S(F_1, F_2)$. It consists of all edges between vertices of F_1 and F_2 having weight w. If a fixed component F splits into F_1 and F_2, edge sets connected to F also split. Consider such an edge set $E_S(F, F')$. Then every edge $e \in E_S(F, F')$ is put either into $E_S(F_1, F')$ or $E_S(F_2, F')$ depending on whether this edge connects F' to F_1 or F_2. Note, that since there are at most $n-1$ fixed components in a layer, the number of created edge sets is upper bounded by $2n$.

The Realms. Up to now we have organized the graph G in fixed components and edge sets in each layer. We still need to introduce some further structure into the layers. We arrange the vertices of G in each layer into realms. The crucial property of realms is that in layer w there may be only edges with weight $w+1$ or larger between vertices which are in separate realms.

To implement the division into realms, we introduce the separating operation, which is applied in every round to every layer and realm separately. In layer w and realm R it looks for a maximum set of vertices V', which has only edges with weight $w+1$ or larger to the rest of vertices in R. Then a new realm is created and the vertices in V' are moved to it. So, the separating operation finds all vertex sets which can be moved to a separate realm.

Fixed components are preserved when vertices are moved to a new realm, i.e. if two vertices v_1 and v_2 have been in a fixed component in realm R then after they are placed in another realm they remain in one fixed component. This requires creating new fixed components in new realms. Analogously, if two vertices have been in separate fixed components in R then they are placed in separate fixed components in the new realm.

The following lemma states a crucial property of the separating operation.

Lemma 1. *Assume that the separating operation is applied in layer w. If a vertex set is put in a new realm and this causes an edge set $E_S(F_1, F_2)$ to be split into two parts E_{S1} and E_{S2} contained in two realms, then only one of E_{S1} and E_{S2} has edges with weight w.*

Interaction between layers. We will now examine the interactions between distinct layers, the fixed components, edge sets and realms on them. We want to show that these interactions follow certain rules and that a certain property (as expressed by Corollary 1) is always fulfilled in a layer.

Lemma 2. *Between fixed components in layer w contained in the same realm there cannot be any edges with weight smaller than w. Each fixed component contains exactly one connected part of $\mathcal{M}_{\mathrm{ALG}}$.*

Corollary 1. *The* RandMST *algorithm uses at most one edge of one edge set in $\mathcal{M}_{\mathrm{ALG}}$.*

The corollary states the most important fact for the analysis of the competitive ratio of RandMST.

Splits and opt*'s cost.* We want to establish a bound between the number of operations opt performs on some layer w and the number of fixed component splits which have occurred on this layer.

Lemma 3. *Let s_w be the number of fixed component splits in layer w during the whole execution of an input sequence. Let $\#E_w(G)$ be the number of edges having weight w in the graph G before round 1. Then* opt *has a cost of at least $s_w - \#E_w(G)$ in layer w.*

By the last lemma, nearly every fixed component split (except for n splits for the whole execution) can be mapped to a round which causes a cost of 1 to OPT. This mapping is obviously injective. If we can map RANDMST's costs to fixed component splits so that each split receives at most $\mathcal{O}(n \log n)$ cost in expectation, then we can easily conclude that the expected competitive ratio of RANDMST is $\mathcal{O}(n \log n)$. We will call this mapping an assignment of costs, and introduce a cost assignment scheme, which assigns RANDMST's costs to fixed component splits.

The cost assignment scheme. Every time a split of a fixed component F into F_1 and F_2 occurs, we assign all created edge sets to this split. This also includes edge sets which are divided in two by the split. This means that an edge set, which has previously been assigned to some split s be assigned to the split of F now. This operation can only decrease the number of edge sets assigned to any fixed component s. Since a fixed component split can create at most $2n$ edge sets, at most $2n$ edge sets are assigned to a split. We still have to bound the cost of RANDMST on one edge set, i.e. bound the number of edge increases in an edge set which causes RANDMST to change an edge in $\mathcal{M}_{\mathrm{ALG}}$.

Consider the way RANDMST chooses a new edge as an alternative for an edge e used before in layer w. This new edge is chosen from the whole alternative set uniformly at random. This alternative set is at least as large as the number of edges with weight w in the current edge set. What is important, is that each of the edges in the current edge set is chosen with the same probability. Thus, even if the adversary knows the code of RANDMST and the probability distribution used by it, it can have no knowledge which particular edge is used within an edge set in $\mathcal{M}_{\mathrm{ALG}}$. On the other hand, by Corollary 1 we know that at most one edge out of an edge set is used in $\mathcal{M}_{\mathrm{ALG}}$.

Let p_{E_S} describe the probability that an edge out of the edge set E_S is used in $\mathcal{M}_{\mathrm{ALG}}$. Let $\#E_S^w$ describe the number of edges with weight w in E_S. Assume that we are now increasing the weight of an edge in edge set E_S. Then, the probability of increasing the weight of an edge which is in $\mathcal{M}_{\mathrm{ALG}}$ is exactly $p_{E_S} \cdot 1/\#E_S^w$. We can upper bound $p_{E_S} \leq 1$. Furthermore, we know that the probability of increasing the weight of an edge in $\mathcal{M}_{\mathrm{ALG}}$ is equal to the expected cost of RANDMST, since RANDMST's cost is either 0 or 1.

To bound the expected cost of RANDMST on an edge set E_S situated in layer w, we only look at requests in the input sequence which increase the weight of an edge set. Each of these requests decreases the number of edges with weight w in E_S by one. What is important and follows from previous considerations, is that the number of edges with weight w in an edge set in layer w never increases after it has been created. So, the expected cost of RANDMST on E_S is then at most $\frac{1}{x} + \frac{1}{x-1} + \ldots + 1$, where x denotes the number of edges with weight w at the moment of the creation of E_S. This value is equal to $\Theta(\log n)$, since we can upper bound $x \leq n^2$.

This cost assignment scheme assures that every change of edges in $\mathcal{M}_{\mathrm{ALG}}$ producing cost is assigned to one edge set and, on the other hand, this edge set

is assigned to a fixed component split. From the fact, that each fixed component split is assigned at most $\mathcal{O}(n)$ edge sets and that each of these edge sets receives an expected cost of $\mathcal{O}(\log n)$ we can easily conclude that the expected competitive ratio of RANDMST is $\mathcal{O}(n \log n)$.

References

1. Pankaj K. Agarwal, David Eppstein, Leonidas J. Guibas, and Monika R. Henzinger. Parametric and kinetic minimum spanning trees. In *FOCS '98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, page 596, Washington, DC, USA, 1998. IEEE Computer Society.
2. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
3. Bernard Chazelle. A minimum spanning tree algorithm with inverse-ackermann type complexity. *J. ACM*, 47(6):1028–1047, 2000.
4. David Cheriton and Robert Endre Tarjan. Finding minimum spanning trees. In *SIAM Journal of Computing*, volume 5, 1976.
5. Francis Chin and David Houck. Algorithms for updating minimal spanning trees. In *Journal of Computer and System Sciences*, volume 16, pages 333–344, 1978.
6. Marek Chrobak, Lawrence L. Larmore, Carsten Lund, and Nick Reingold. A better lower bound on the competitive ratio of the randomized 2-server problem. *Information Processing Letters*, 63(2):79–83, 1997.
7. David Eppstein. Spanning trees and spanners. Technical Report ICS-TR-96-16, 1996.
8. David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.
9. Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 252–257, New York, NY, USA, 1983. ACM Press.
10. Monika Rauch Henzinger and Valerie King. Maintaining minimum spanning trees in dynamic graphs. In *ICALP '97: Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, pages 594–604, London, UK, 1997. Springer-Verlag.
11. J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematics Society*, volume 7, pages 48–50, 1956.
12. Dimitrios E. Pendarakis, Sherlia Shi, Dinesh C. Verma, and Marcel Waldvogel. Almi: An application level multicast infrastructure. In *3rd USENIX Symposium on Internet Technologies and Systems*, pages 49–60, 2001.
13. Seth Pettie and Vijaya Ramachandran. An optimal minimum spanning tree algorithm. *J. ACM*, 49(1):16–34, 2002.
14. R.C. Prim. Shortest connection networks and some generalizations. In *Bell System Technical Journal*, volume 36, pages 1389–1401, 1957.
15. P.-J. Wan, G. Calinescu, X.-Y. Li, and O. Frieder. Minimum-energy broadcasting in static ad hoc wireless networks. In *Wireless Networks, Volume 8, Issue 6*, volume 8, pages 607–617, 2002.
16. A. Young, J. Chen, Z. Ma, A. Krishnamurthy, L. Peterson, and R.Y. Wang. Overlay mesh construction using interleaved spanning trees. In *IEEE INFOCOM*, pages 396–407, 2004.