# Minimum Spanning Tree Verification Under Uncertainty

Thomas Erlebach and Michael Hoffmann(✉)

Department of Computer Science, University of Leicester, Leicester, UK
{te17,mh55}@mcs.le.ac.uk

**Abstract.** In the *verification under uncertainty* setting, an algorithm is given, for each input item, an *uncertainty area* that is guaranteed to contain the exact input value, as well as an assumed input value. An *update* of an input item reveals its exact value. If the exact value is equal to the assumed value, we say that the update *verifies* the assumed value. We consider verification under uncertainty for the minimum spanning tree (MST) problem for undirected weighted graphs, where each edge is associated with an uncertainty area and an assumed edge weight. The objective of an algorithm is to compute the smallest set of updates with the property that, if the updates of all edges in the set verify their assumed weights, the edge set of an MST can be computed. We give a polynomial-time optimal algorithm for the MST verification problem by relating the choices of updates to vertex covers in a bipartite auxiliary graph. Furthermore, we consider an alternative uncertainty setting where the vertices are embedded in the plane, the weight of an edge is the Euclidean distance between the endpoints of the edge, and the uncertainty is about the location of the vertices. An update of a vertex yields the exact location of that vertex. We prove that the MST verification problem in this vertex uncertainty setting is NP-hard. This shows a surprising difference in complexity between the edge and vertex uncertainty settings of the MST verification problem.

## 1 Introduction

In this paper we consider settings where a solution to a combinatorial problem needs to be computed and where the input data of the problem might change over time. We assume that the data cannot change arbitrarily and thus the new data is guaranteed to be somewhat close to the old data, represented by an *uncertainty area* for each input data item. The operation of checking the current exact value of an input item, which we also refer to as an *update*, may be expensive, so we want to avoid applying it to all input data items. Moreover, it is possible that the input data is stable and has not changed. One would then like to *verify* for a small set of input data that their values have not changed so that a solution to the combinatorial problem can be calculated based on the verified input data and the given uncertainty areas. We refer to problems of this kind as *verification under uncertainty*.

D. Kratsch and I. Todinca (Eds.): WG 2014, LNCS 8747, pp. 164–175, 2014.
DOI: 10.1007/978-3-319-12340-0_14

In practice, such settings arise naturally, e.g., when maintaining an optimal routing structure in wireless networks with nodes that are generally static but may occasionally move within a limited area. If the exact node positions were known at some point in the recent past, the possible node positions at the current time are known to lie in uncertainty areas that are limited regions around the original positions of the nodes. One can also imagine scenarios in which nodes automatically send a notification message if their location changes by more than a certain threshold. In such scenarios, if none of the nodes has sent a notification since the last determination of exact positions, the area within the threshold distance of the previous location of a node becomes its uncertainty area. As the size of such uncertainty areas is independent of the time that has elapsed since the last determination of the exact position, frequent requests to compute a solution are better addressed in the verification setting. Finally, in a network setting where edge weights represent link congestion, we may again have scenarios where exact weights were known at a point in the past and the current edge weights are guaranteed to lie in certain intervals represented by uncertainty areas. These scenarios have in common that it is possible to obtain the exact current data (node positions or link congestion values) at some cost, and one is interested in being able to compute a solution after verifying only for a small subset of the input data that the data has not changed.

In this paper, we consider the minimum spanning tree (MST) verification problem under uncertainty. The MST is one of the most fundamental graph structures and relevant in many application areas, including routing in wireless ad-hoc networks. We study two uncertainty settings: In the *edge uncertainty* setting, each edge $e$ has an uncertainty area $A_e$ that is guaranteed to contain its current weight, and an update of the edge reveals its exact current weight. In the *vertex uncertainty* setting, the graph is a complete graph embedded in the plane and the weight of an edge is the Euclidean distance between its endpoints. The uncertainty is in the positions of the nodes, and an update of a node reveals its exact current position. In both settings, the goal is to compute a minimum set of updates such that, if these updates verify the expected input data, the edge set of an MST can be calculated.

**Our Results.** We obtain the following results for MST verification under uncertainty:

- For MST verification under edge uncertainty, provided that the uncertainty areas are open sets or trivial (i.e., contain only one value), we obtain a polynomial-time optimal algorithm by relating sets of updates to vertex covers in a bipartite auxiliary graph that is constructed by adapting a witness set algorithm.
- We show that MST verification under vertex uncertainty is NP-hard even if the uncertainty areas are trivial or open disks. The proof is by reduction from the vertex cover problem for planar graphs with maximum degree 3.
- As an auxiliary result used in the NP-hardness proof, we show that every planar graph of maximum degree 5 can be represented by a unit disk graph (after introducing degree-two vertices on each edge of the planar graph). Although

embeddings of planar graphs as unit disk graphs have been used in the past for NP-hardness proofs, our embedding of planar graphs of degree 5 may be of independent interest.

Our work contributes to the wider research area of computing under uncertainty that studies the problem of minimizing the cost of obtaining exact input values in settings where some of the input data is uncertain. Traditional research in optimization assumes that all input data is given precisely. In cases where the input data is not known precisely (e.g., only a probability distribution for the input data values is known), a substantial amount of research in areas such as stochastic programming or robust optimization has focussed on computing solutions that are good (e.g., in expectation, with high probability, or in the worst case) no matter what the exact values of the input data are. The area of computing under uncertainty approaches problems with uncertain input data from a different angle by assuming that an algorithm can obtain the exact value of an input data item at a certain cost (by performing an update), and aiming to minimize the cost of updates while guaranteeing that an exact solution can be computed.

Work in computing under uncertainty falls in three main categories: In the *adaptive online* setting an algorithm initially knows only the uncertainty areas and performs updates one by one (determining the next update based on the information from previous updates) until it has obtained sufficient information to determine a solution. Algorithms are typically evaluated by competitive analysis, comparing the number of updates they make with the minimum number of updates that, in hindsight, would have been sufficient to determine a solution (referred to as the offline optimum). In the *non-adaptive online* setting an algorithm is also given only the uncertainty areas initially, but it must determine a set $U$ of updates such that after performing all updates in $U$ it is guaranteed to have sufficient information to determine a solution. Finally, there is the *verification* setting that was already described above. It is worth noting that the optimal update set of the *verification* setting is also the offline optimum of the adaptive online setting. Therefore, algorithms solving the verification problem are also useful for the experimental evaluation of algorithms for the adaptive online setting.

**Related Work.** Kahan [7] presented a model for handling imprecise but updateable input data. He demonstrated his model on a set of real numbers where instead of the precise value of each number an interval was given. That interval when updated reveals that number. The aim is to determine the maximum, the median, or the minimal gap between any two numbers in the set, using as few updates as possible. His work included a competitive analysis for this type of online algorithm, where the number of updates is measured against the optimal number ($OPT$) of updates. For the problems considered, he presented online algorithms with optimal competitive ratio. Feder et al. [4] studied the problem of computing the value of the median of an uncertain set of numbers up to a certain tolerance. Applications of uncertainty settings can be found in many different areas including databases, geometry and structured data such as graphs. The work presented in this paper mainly concerns the latter two areas.

Bruce et al. [1] studied geometric uncertainty problems in the plane. Here, the input consists of points in the plane and the uncertainty information is for each point of the input an area that contains that point. They presented algorithms with optimal competitive ratio for the maximal point problem and the convex hull problem. Both algorithms are based on a more general technique called a *witness set algorithm* that was introduced in their paper.

Examples of uncertainty applications to graphs include [3], where Feder et al. investigated shortest paths on graphs with uncertain edge weights. They allowed a precision factor limiting the deviation from the actual shortest path and presented results on adaptive as well as non-adaptive updates.

In [2], Erlebach et al. studied the adaptive online setting for MST under two types of uncertainty: the edge uncertainty setting, which is the same as the one considered by Feder et al. [3], and the vertex uncertainty setting. In the latter setting, all vertices are points is the plane and the graph is a complete graph with the weight of an edge being the distance between the vertices it connects. The uncertainty is given by areas for the location of each vertex. For both settings, Erlebach et al. presented algorithms with optimal competitive ratio for the MST under uncertainty. The competitive ratios are 2 for edge uncertainty and 4 for vertex uncertainty, and the uncertainty areas must satisfy certain restrictions (which are satisfied by, e.g., open and trivial areas in the edge uncertainty case). A variant of computing under uncertainty where updates yield more refined estimates instead of exact values was studied by Gupta et al. [6].

A different setting of the MST under vertex uncertainty was studied by Kamousi et al. [8]. They assume that point locations are known exactly, but each point $i$ is present only with a certain probability $p_i$. They show that it is #P-hard to compute the expected length of an MST even in 2-dimensional Euclidean space, and provide a fully polynomial randomized approximation scheme for metric spaces.

**Paper Outline.** In Sect. 2 we give formal definitions and preliminaries. Section 3 presents our optimal algorithm for MST verification under edge uncertainty. In Sect. 4 we give the proof of NP-hardness for MST verification under vertex uncertainty.

Some proofs of Sects. 3 and 4 are omitted.

## 2 Preliminaries

Within the wider field of problems under uncertainty we consider the minimum spanning tree problem for graphs under uncertainty. We consider undirected weighted graphs $G = (V, E)$ under two different types of uncertainty. In an *edge uncertainty* graph, the weight $W_e$ of an edge $e$ might not be known exactly, but instead a set $A_e$ of possible values of $W_e$ is given. We let $W$ be the set that contains for each $e \in E$ the exact weight $W_e$, and $\mathcal{A}$ the set that contains for each $e \in E$ its uncertainty area $A_e$. We refer to $\mathcal{A}$ as the *areas of uncertainty.*

In this type of uncertainty, the update of an edge $e$ reveals its exact weight $W_e$. This effectively changes the uncertainty area $A_e$ to the singleton set containing just $W_e$ (we call such a set *trivial*). An instance of the MST-EDGE-UNCERTAINTY problem is given by $(G, W, \mathcal{A})$.

In the second type of uncertainty, we consider *vertex uncertainty* graphs. Here the graph is a complete graph embedded in the plane. The weight of each edge is given by the distance of the vertices that it connects. For each vertex $v \in V$ the location of $v$ might not be known exactly, but instead a set $A_v$ of possible locations of $v$ is given, and $\mathcal{A}$ is the family of all these uncertainty areas $A_v$. An update of a vertex $v$ reveals the exact location $P_v$ of $v$. An instance of the MST-VERTEX-UNCERTAINTY problem is given by $(G, P, \mathcal{A})$, where $P$ is the set containing the precise vertex location $P_v$ for each $v \in V$.

In the online setting, the precise information ($W$ for edge uncertainty and $P$ for vertex uncertainty graphs) is not known to the algorithm; the algorithm has to request updates until $\mathcal{A}$ is precise enough to allow the calculation of an MST of $G$. In the verification setting, the sets $W$ or $P$ respectively are given to the algorithm. This additional information is not used to calculate an MST of $G$ directly, but it is used to determine which updates should be made so that an MST of $G$ can be calculated based on the updated areas of uncertainty. A set of updates that reveals enough information so that an MST of $G$ can be calculated is called an *update solution*, and the set of all update solutions is denoted by $S$. For a given instance of a problem, we denote the size of the smallest update solution by $OPT$. In the verification setting, the goal of an algorithm is to calculate an update solution of size $OPT$.

In the remainder of the paper we will use the following notion: $G = (V, E)$ is the weighted undirected graph for which an MST should be found; we say $\mathcal{U}$ is an *uncertainty graph of* $G$ if it consists of the same edges and vertices as $G$, but only contains the uncertain information as specified by $\mathcal{A}$. $S$ is the set of update solutions, and $OPT$ is the size of the smallest element of $S$.

In Fig. 1, the left-hand side shows a graph $G$, and an uncertainty graph $\mathcal{U}$ of $G$ is given on the right. $G$ has two minimal spanning trees, with edge sets $\{b, c, d, f\}$ and $\{b, d, e, f\}$. None of them can be calculated based on the information of $\mathcal{U}$ alone, as for example the weight of the edge $a$ could be smaller than that of $b$. The set of update solutions is $S = \{\{a, b, e\}, \{a, e\}, \{b, e\}\}$, and both $\{a, e\}$ and $\{b, e\}$ have minimum size. Thus, in this example $OPT$ is 2.

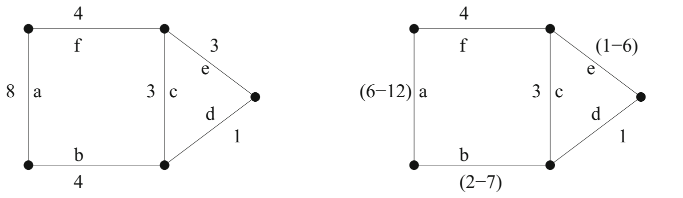

**Fig. 1.** Example of graph $G$ (left) and edge uncertainty graph $\mathcal{U}$ (right)

## 3 Verification Under Edge Uncertainty

In this section we give a polynomial-time algorithm for MST-EDGE-UNCERTAINTY. We assume that the uncertainty areas of the edge weights are trivial or open areas. The first phase of the algorithm is based on the algorithm U-RED that was presented in [2] for the adaptive online version of the MST under edge uncertainty problem. Adopting the principles of this online algorithm to the verification setting is non trivial. Roughly speaking, the algorithm U-RED repeatedly identifies an edge $e$ that, based on the current areas of uncertainty, may or may not be in an MST. It then identifies at most two edges such that without updating any of the two edges the edge $e$ can neither be included nor excluded from any MST (these two edges thus form a so-called witness set), and updates both of the edges.

Whereas U-RED updates both of these edges, we utilize the additional information of the precise values that are available in the verification setting. It turns out that with this information we can always arrive at one of the following two cases: (1) There is a single edge $f$ such that, without updating $f$, the edge $e$ can neither be included nor excluded from any MST. In this case, we record that $f$ is needed to be in any update solution and simulate for U-RED the update of $f$. (2) There is a choice of edge sets whose updates can determine whether $e$ is in an MST or not. In this case, we record the choice. We can prove that this can only happen when $e$ is not in any MST, so we remove that edge and continue the simulation of U-RED.

After the simulated run of U-RED, we have established a set of updates that are common among all update solutions and we have also recorded a set of choices. We show that each choice is between a single edge and a single set of edges. We also show that the set of choices has additional properties that allow us to model it as a bipartite graph in such a way that a minimum vertex cover of the bipartite graph yields a minimum set of updates to cover all choices. Together with the already established set of common updates this gives a minimum update solution for MST-EDGE-UNCERTAINTY.

**Theorem 1.** *There is a polynomial-time algorithm that computes an optimal update solution for instances of* MST-EDGE-UNCERTAINTY *where the uncertainty areas are trivial or open areas.*

In the remainder of this section we present the algorithm and prove its correctness, thus establishing Theorem 1. The algorithm runs in three phases. In the first phase, two sets $A$ and $R$ are constructed. The set $A \subseteq E$ is the set of edges that are common to all update solutions, i.e., $A = \bigcap_{s \in S} s$. (Recall that $S$ denotes the set of all update solutions.) The set $R \subseteq E \times \mathcal{P}(E)$ consists of pairs $(d, B)$ with $d \in E$ and $B \subseteq E$. Each pair $(d, B) \in R$ represents a choice with the property that every update solution must contain $d$ or all elements of $B$. In addition, $R$ is of the form that any combination of choosing either the single edge or the set of edges together with the set $A$ is an update solution. As we will refer to these properties later on, we state them formally as follows.

*Property 1.* The sets $A$ and $R = \{(d_1, B_1), \ldots, (d_n, B_n)\}$ satisfy the following properties:

- **p1:** $A = \bigcap_{s \in S} s$
- **p2:** If $s$ is an update solution, then for all $1 \leq i \leq n$ we have $d_i \in s$ or $B_i \subseteq s$.
- **p3:** $S' = \{A \cup \{d_i | i \in I\} \cup \bigcup\limits_{j \in J} B_j \mid I, J \text{ form a partition of } \{1, \ldots, n\}\}$ is a set of update solutions.

As a consequence of p1–p3, for every update solution $s$ there exists $s' \in S'$ such that $s' \subseteq s$.

From the outset it is not clear that a set $R$ satisfying p1–p3 exists. We will show that it does and how to construct it. In the second phase, redundant choices in $R$ will be removed without altering the properties of $R$. In the third and final phase, we model the choice selection for $R$ as a vertex cover problem in a bipartite graph. We will show that an optimal solution to the latter problem results in an update solution for the MST verification problem of minimum size.

**Phase 1.** The aim of this phase is to establish the sets $A$ and $R$ described above. The algorithm used in this phase is based on (a simulation of) the online algorithm U-Red presented in [2]. The significant changes include: (1) The online algorithm U-Red restarts after each update, whereas our algorithm avoids restarts and sorts the updated edges back into the running process. (2) When the online algorithm U-Red updates the edges in a witness set, we utilize the information of the exact weights of the edges involved and determine the appropriate contribution to the sets $A$ and $R$ instead. The resulting Algorithm Phase1 is given in Fig. 2. It uses the notation of the following definitions.

**Definition 1.** *For an edge $e$ in an edge-uncertainty graph, we denote the actual weight of the edge by $W_e$ and the upper limit of $A_e$ by $U_e =$ lim sup $\{a \mid a \in A_e\}$ and the lower limit of $A_e$ by $L_e =$ lim inf $\{a \mid a \in A_e\}$.*

Note that, as edges are updated in the algorithm, the values for $U_e$ and $L_e$ for an edge $e$ may change. In particular, after updating the edge $e$ we have that $L_e = W_e = U_e$.

**Definition 2.** *The order by which the edges are sorted in Algorithm* Phase1 *is as follows: Let $\mathcal{U}$ be an edge-uncertainty graph and let $e, f$ be two edges of $\mathcal{U}$. We define $e < f$ if $L_e < L_f$ or ($L_e = L_f$ and $U_e < U_f$). Edges with the same upper and lower weight limit are ordered arbitrarily.*

**Definition 3.** *Let $C$ be a cycle in $\mathcal{U}$ and $e \in C$. The edge $e$ is said to be always maximal in $C$ if for all possible weights that are consistent with the uncertainty areas given by $\mathcal{U}$ the weight of $e$ is maximal among the weights of all edges in $C$.*

Note that updating edges in $\mathcal{U}$ only reduces the options for the edge weights. Hence, an always maximal edge in $C$ remains an always maximal edge in $C$ after updating arbitrary elements of $\mathcal{U}$.

```
01 Create a list $L$ of all edges in the order of Definition 2 from low to high
02 Let $\Gamma$ be $\mathcal{U}$ without any edge
03 while $L$ is not empty do
04     add the head of $L$ to $\Gamma$
05     remove the head of $L$
06     if $\Gamma$ has a cycle $C$ then
07         case (a): $C$ contains an always maximal edge $e$.
08             delete $e$ from $\Gamma$
09         case (b): There exists $e \in C$ whose update must be in any update solution.
10             update $e$, remove $e$ from $\Gamma$, and add $e$ to $A$
11             sort $e$ back into $L$
12         case (c): There is a choice of updates that establish an edge as
13                 an always maximal edge in the cycle $C$.
14             add the choice to $R$
15             delete the always maximal edge from $\Gamma$
16     end if
17 end while
```

**Fig. 2.** Algorithm Phase1

Before showing that there exists (in line 15) a unique always maximal edge, and that the sets $A$ and $R$ are built correctly, we establish the following lemma which gives a locality property: Updates required on the basis of just one cycle will never be made redundant by other updates or cycles in the graph.

**Lemma 1.** *During the run of the algorithm, when a cycle $C$ is closed, let $e$ be a non-trivial edge in $C$. If updating a set $U \subseteq E - \{e\}$ does not determine whether $e$ is always maximal in $C$, then updating $U$ will also not verify that $e$ is always maximal in any other cycle.*

Once a cycle in $\Gamma$ is formed during the run of the algorithm, different actions are taken. We will show that the cases listed in the algorithm cover all possibilities and that the sets $A$ and $R$ are built correctly.

The first check after a cycle $C$ in $\Gamma$ is formed is whether there exists, according to the current uncertainty information, an edge in $C$ that is always maximal. If such an edge exists, the algorithm executes case (a) and the edge is deleted from $\Gamma$, no update is made, and the sets $A$ and $R$ stay unaltered.

If case (a) does not apply, let $h$ be an edge in $C$ with maximum upper limit $U_h$. Note that $h$ must be non-trivial (otherwise case (a) would apply). There are four possible cases for how the actual weight $W_h$ relates to the weights and limits of other edges in the cycle $C$. **Case 1.** If $W_h$ is not maximal among the actual weights of all edges in $C$, then $h$ needs to be updated in any update solution (the algorithm executes case (b)). **Case 2.** If $W_h$ is maximal amongst the actual weights of edges in $C$ and there exists an $f \in C$ with $U_f > W_h$, then $f$ needs to be updated in any update solution (the algorithm executes case (b)). **Case 3.** If $W_h$ is maximal amongst the actual weights of edges in $C$ and there exists an $f \in C$ such that $W_f > L_h$, then $h$ needs to be updated in any update solution (the algorithm executes case (b)). **Case 4.** If Cases 1–3 do not apply, every update solution must contain $h$ or all edges of the set $B = \{c \in C - \{h\} \mid U_c > L_h\}$, so the algorithm executes case (c).

*Remark 1.* In the situation of Case 4, the edge $h$ is greater in the order of edges used by the algorithm than any other edge in the cycle $C$ (i.e., $h > c$ for all $c \in C - \{h\}$) and hence was the edge that closed the cycle.

From the above we can conclude that the set $A$ only contains updates that are in any update solution, that for all $(d, B) \in R$ an update solution must include the edge $d$ or all edges in $B$, and that any set of edges containing all elements of $A$ and from every pair $(d, B) \in R$ at least $d$ or all edges in $B$ is an update solution. This shows that $A$ and $R$ satisfy properties p2 and p3.

Since for every pair $(d, B) \in R$ the edge $d$ is not in $B$ (see also Lemma 2 below), there exists for every $g \notin A$ an update solution not containing $g$. This shows that $A$ is the intersection of all update solutions, establishing that p1 is satisfied as well. Before tidying up $R$ in phase 2 (in a way that maintains p1–p3), we establish an additional property of $R$ that will be used in phase 3 to build a bipartite graph.

**Lemma 2.** *Let $(d, B)$ and $(d', B')$ in $R$. Then $d \notin B'$.*

*Proof.* Assume there exist $(d, B)$ and $(d', B')$ in $R$ with $d \in B'$. When a pair $(d, B)$ is added to $R$, the edge $d$ is deleted from $\Gamma$ and hence will not be part of any pair that is added to $R$ later. So for $d$ being an element of $B'$, the pair $(d', B')$ must have been added to $R$ before $(d, B)$. By Remark 1, $d' \leq d$. Considering that $d \in B'$ we also have by the same remark that $d < d'$, which gives a contradiction. □

**Phase 2.** As the sets $A$ and $R$ are built up simultaneously, it is possible that for a pair $(d, B)$ in $R$ some edges in $B$ are added to $A$ later on in the run of the Algorithm Phase1. Since the edge $d$ is deleted from $\Gamma$ when a pair $(d, B)$ is added to $R$, the edge $d$ can never be added to $A$.

In this short phase, $R$ is tidied up by the following steps: For every $(d, B) \in R$ all elements of $B$ that are also in $A$ will be removed from $B$. Where, as a result, $B$ becomes empty, the entire pair $(d, B)$ is removed from $R$. Formally $R$ is replaced by $\{(d, B) \mid \exists (d, B') \in R, B = B' - A, B \neq \emptyset\}$. This does not affect the properties p1–p3 of Property 1.

**Phase 3.** In this final phase, an optimal update set is calculated from the sets $A$ and $R$. As stated in p3 of Property 1, a set $S'$ of update solutions can be formed from $A$ and $R$. An update solution with minimum size amongst them can be established by modelling the choices as a vertex cover problem in a bipartite graph. We then show that there is no update solution with fewer updates. Recall the notation of p3: $R = \{d_1, B_1), \ldots, (d_n, B_n)\}$ and $S' = \{A \cup \{d_i | i \in I\} \cup \bigcup_{j \in J} B_j \mid I, J \text{ partition of } \{1, \ldots, n\}\}$. In phase 2, any overlap between elements of $A$ and elements appearing in the pairs of $R$ was removed from $R$. So, to find an element of $S'$ with minimum size it is enough to find an element of minimum size in $R' = \{\{d_i | i \in I\} \cup \bigcup_{j \in J} B_j \mid I, J \text{ partition of } \{1, \ldots, n\}\}$, as $A$ and the elements of $R'$ are disjoint.

We now create a bipartite graph $G'$ for which any element of $R'$ is a vertex cover and any vertex cover must contain one element of $R'$ as a subset. Loosely speaking, every edge of the uncertainty graph $\mathcal{U}$ occurring inside any element of $R$ is a node in $G'$. For every choice $(d, B) \in R$, the node in $G'$ corresponding to $d$ is connected to the nodes in $G'$ corresponding to the elements of $B$.

Let $G' = (V', E')$ be an undirected graph with $V' = \{d_1, \ldots, d_n\} \cup B_1 \cup \cdots \cup B_n$ and $E = \{(d_i, b) \mid b \in B_i, 1 \le i \le n\}$. By Lemma 2, the set $\{d_1, \ldots, d_n\}$ and $B_1 \cup \cdots \cup B_n$ are disjoint. As every edge in $G'$ connects an element from $\{d_1, \ldots, d_n\}$ to an element of $B_1 \cup \cdots \cup B_n$, $G'$ is a bipartite graph.

Every element of $R'$ contains, for every $i$, the edge $d_i$ or all elements of $B_i$. Therefore, every element of $R'$ is a vertex cover for $G'$. Similarly if a vertex cover of $G'$ does not include $d_i$ for any $i$, then it must include all elements of $B_i$ and hence it must contain an element of $R'$ as a subset. Thus, a minimum vertex cover $r^*$ of $G'$ is also an element of $R'$ with minimal size. Furthermore, $A \cup r^*$ is an element of minimum size in $S'$. By Property 1, for every update solution $s$ there exists an $s' \in S'$ such that $s' \subseteq s$. So $A \cup r^*$ is of minimum size amongst all update solutions, and $OPT = |A \cup r^*|$.

Noting that the minimum vertex cover problem is polynomial for bipartite graphs, it is not difficult to show that the algorithm runs in polynomial time. Hence, a minimal update solution (of size $OPT$) for MST-EDGE-UNCERTAINTY under the restriction to open and trivial areas can be computed in polynomial time. This completes the proof of Theorem 1.

Furthermore, we note that the algorithm can be extended to a version of the problem where each edge $e$ has an arbitrary update cost $c_e > 0$ and the goal is to minimize the total cost of the update solution. The approach is the same, except that the vertex cover problem in the bipartite auxiliary graph needs to be solved as a minimum-weight vertex cover problem.

**Theorem 2.** *For the* MST-EDGE-UNCERTAINTY *problem with arbitrary positive update costs and under the restriction to open or trivial areas, an optimal update solution can be computed in polynomial time.*

## 4 Verification Under Vertex Uncertainty

In this section we prove that MST-VERTEX-UNCERTAINTY is NP-hard. The proof uses a reduction from the vertex cover problem in planar graphs of maximum degree 3, which was shown to be NP-complete in [5]. In the reduction we use the following embedding result.

**Theorem 3.** *Let $G = (V, E)$ be a planar graph of maximum degree 5 with $n$ vertices. Then there exists a value $s > 0$ and an embedding of $G$ such that*

- *vertices are mapped to integer coordinates in an $n$ by $n$ grid,*
- *edges are mapped to non-crossing paths (consisting of straight line segments and circular arcs),*

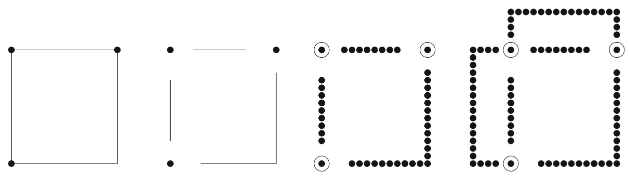

**Fig. 3.** Transformation of the graph $G$

- *the length of each path is polynomial in $n$,*
- *$1/s$ is polynomial in $n$,*
- *in the disk of radius $s$ around each vertex, all edges are equally spaced straight lines,*
- *everywhere else the edges are at least $s$ apart,*
- *the embedding can be constructed in polynomial time.*

We give an outline of the proof as the detailed proof is somewhat technical. The starting point is an arbitrary planar graph of maximum degree 3. Finding a minimum vertex cover for such graphs is NP-hard [5]. We transform this graph to an instance of MST-VERTEX-UNCERTAINTY by the following three steps (illustrated in Fig. 3). The steps are given here in an order that reflects the motivation for the steps, but for technical reasons the order will be different in the actual reduction.

**Step 1: Create an uncertainty problem.** After embedding the graph in the plane, we shorten each edge so that instead of connecting two vertices, it falls short at both ends. At one end it leaves a gap of $1 + 5\epsilon$ to the vertex, and at the other a gap of $1 + 8\epsilon$ to the other vertex. Finally, each vertex is replaced by an uncertainty area that is a disk of radius $2\epsilon$ around the original location of the vertex. So, each edge has at one end a gap between $1 + 3\epsilon$ and $1 + 7\epsilon$ and at the other end a gap between $1 + 6\epsilon$ and $1 + 10\epsilon$. If one wants to know for each edge which end has the smaller gap, one has to update at least one of the vertices that it connected originally. If the precise location of each vertex lies at the center of its uncertainty disk, then updating either end vertex of an edge will determine at which end the edge has a smaller gap. Thus, finding the smallest set of vertices that needs to be updated to determine for each edge at which end it has the smaller gap is equivalent to finding a minimum vertex cover of the original graph.

**Step 2: Create a vertex uncertainty graph.** To convert the graph of step 1 to a vertex uncertainty graph, we replace each edge fragment by a dense sequence of new vertices. The position of these vertices is given exactly (i.e., their uncertainty areas are trivial). The distance of two neighboring vertices is less than 1/2.

**Step 3: Create a minimum spanning tree problem.** To turn the question that asks at which side each former edge has the smaller gap into an MST problem, we place additional vertices such that all original vertices are connected via dense ‘lines’ made out of these new vertices, and the original vertices and the new ‘lines’ form a tree. The gap of such a ’line’ to an original vertex is 1. If all lines can be placed in such a way that all vertices on one line are far away (at least distance 1) from any vertex on another line, solving the MST-VERTEX-UNCERTAINTY problem requires updating for each original edge at least one of its end points. The minimum set of such updates yields a minimum vertex cover of the original graph. In the actual proof we add the auxiliary ‘lines’ of Step 3 already before embedding the graph for Step 1, and the distances mentioned above are scaled down by an appropriate scaling factor.

**Theorem 4.** *Calculating OPT for* MST-VERTEX-UNCERTAINTY *is NP-hard.*

As the exact weight of any edge can be obtained by updating both of its vertices, the polynomial optimal algorithm for MST-EDGE-UNCERTAINTY can be used to obtain a 2-approximation of the MST-VERTEX-UNCERTAINTY problem.

**Acknowledgements.** The second author would like to thank the University of Leicester for supporting this research in granting him academic study leave.

## References

1. Bruce, R., Hoffmann, M., Krizanc, D., Raman, R.: Efficient update strategies for geometric computing with uncertainty. Theor. Comput. Syst. **38**(4), 411–423 (2005)
2. Erlebach, T., Hoffmann, M., Krizanc, D., Mihalák, M., Raman, R.: Computing minimum spanning trees with uncertainty. In: Albers, S., Weil, P. (eds.) STACS. LIPIcs, vol. 1, pp. 277–288. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2008)
3. Feder, T., Motwani, R., O’Callaghan, L., Olston, C., Panigrahy, R.: Computing shortest paths with uncertainty. J. Algorithms **62**(1), 1–18 (2007)
4. Feder, T., Motwani, R., Panigrahy, R., Olston, C., Widom, J.: Computing the median with uncertainty. SIAM J. Comput. **32**(2), 538–547 (2003)
5. Garey, M., Johnson, D.: The rectilinear Steiner tree problem is NP-complete. SIAM J. Appl. Math. **32**(4), 826–834 (1977)
6. Gupta, M., Sabharwal, Y., Sen, S.: The update complexity of selection and related problems. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011). LIPIcs, vol. 13, pp. 325–338. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2011)
7. Kahan, S.: A model for data in motion. In: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC’91), pp. 267–277 (1991)
8. Kamousi, P., Chan, T.M., Suri, S.: Stochastic minimum spanning trees in Euclidean spaces. In: Proceedings of the 27th Annual ACM Symposium on Computational Geometry (SoCG’11), pp. 65–74. ACM (2011)