Journal of Math-for-Industry, Vol.2(2010B-8), pp.183–197

A variation of the minimum spanning tree problem for the application to mathematical OCR

Akio Fujiyoshi and Masakazu Suzuki

Received on September 23, 2010

Abstract. In this paper, we introduce a variation of the minimum spanning tree problem for the application to mathematical OCR. The problem is obtained from the original minimum spanning tree problem by importing the notions of "candidate selection" and "link-label selection." It is shown that the problem is NP-hard. However, we find that, for the application to mathematical OCR, it is sufficient to deal with only a class of graphs that is recursively defined with some graph-rewriting rules. For the class of graphs, it is shown that the problem can be solved in linear-time in the number of vertices of a graph.

Keywords. the minimum spanning tree problem, NP-completeness, treewidth, mathematical OCR, mathematical formula recognition.

1. Introduction

The minimum spanning tree problem (MSTP) is one of the most famous combinatorial problems in algorithmic graph theory. Since MSTP has numerous applications in network design, it has been extensively studied. Polynomial-time algorithms to solve MSTP are well-known such as Kruskal's algorithm [7] and Prim's algorithm [10]. For some specific applications, variations of MSTP have also been studied. Myung, Lee and Tcha [9] introduced the generalized minimum spanning tree problem (GMSTP), where the vertices of a graph are partitioned into clusters and exactly one vertex from each cluster must be connected. Chang and Leu [5] introduced the minimum labeling spanning tree problem (MLSTP), where the edges of a graph are colored and the number of colors of a spanning tree should be minimized. Makino, Uno and Ibaraki [8] introduced the minimum edge-ranking spanning tree problem (MERSTP), that is, the problem of finding a spanning tree of a graph whose edge-ranking is minimum. For the application to mathematical OCR [4], we introduce another variation of MSTP in this paper.

Eto and Suzuki [6] proposed a recognition method for mathematical formulae using "virtual link networks." As shown in Figure 1, a mathematical OCR system constructs a directed acyclic graph (DAG) that expresses possible adjacency connections between bounding boxes from a scanned image and then outputs a recognition result, which forms a tree structure. The idea is that, if we put proper weights on the links representing adjacency connections between bounding boxes, the correct recognition result should corresponds to the minimum spanning tree. The problem of finding the minimum spanning tree of a virtual link network is not as simple as the minimum spanning tree problem of ordinary graphs. An example of a virtual link network is illustrated in Figure 2. There exist several candidates of character recognition for each bounding box: The left-most bounding box has candidates "italic x" and "Greek lowercase Chi"; the second has candidates "number zero" and "Roman o"; the third has a candidate

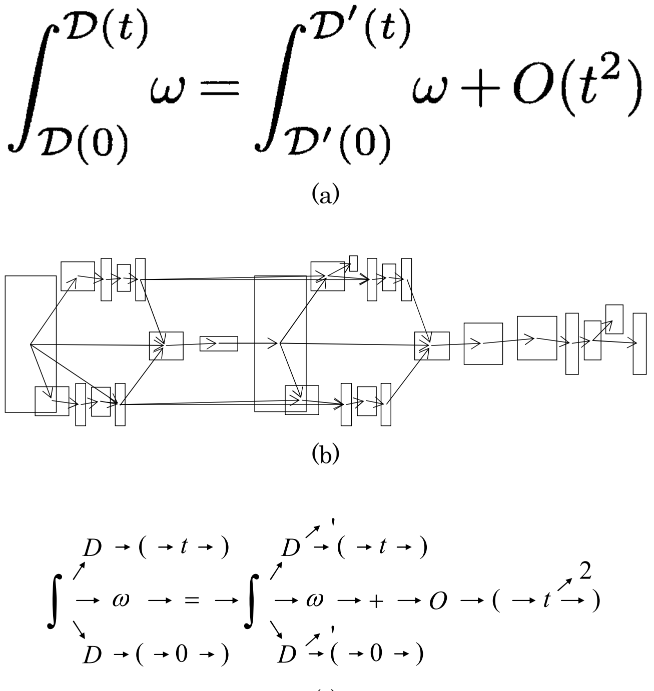


Figure 1: (a) A scanned image, (b) the graph expressing possible adjacency links of bounding boxes, (c) a recognition result.

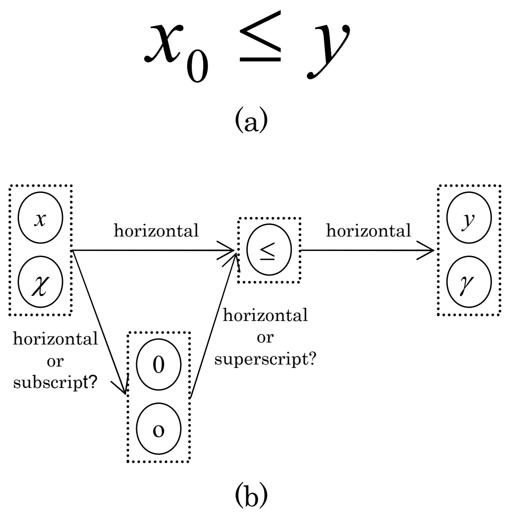


Figure 2: (a) A scanned image and (b) the corresponding virtual link network.

"less than or equal to sign"; and the fourth has "italic y" and "Greek lowercase Gamma." The links are labeled with possible types of adjacency connections: "horizontal", "superscript", "subscript", "upper", "under", and so on. We need to find the minimum spanning tree of a virtual link network not only by selecting character recognition candidates for bounding boxes but also by determining types of adjacency connections between bounding boxes. We call this variation of MSTP "the minimum spanning tree problem with candidate and link-label selections."

In pattern recognition, it is important to study this type of combinatorial problems because the combination of plural types of classifiers is an important issue in this field. For mathematical OCR, we need to combine a classifier for character recognition with a classifier for structure recognition. It is difficult to obtain good recognition results if classifiers are sequentially combined, i.e, executing one classifier first and then putting the result to another classifier. In order to obtain better recognition results, classifiers should be simultaneously combined. We think that, in many situations, simultaneous combinations of classifiers can be formulated as a similar combinatorial problem and solved in a similar way.

Unfortunately, in this paper, it will be shown that the problem is NP-hard. The NP-hardness is proved by reducing the Boolean satisfiability problem (SAT) to the problem. Therefore, it is impossible to solve the problem in polynomial time for the general case unless P=NP.

However, there is a clue to solve the problem. In graph theory, it is known that many NP-hard combinatorial problems on graphs can be solvable in polynomial time if the treewidth of an input graph is bounded by a small number [1, 2, 3]. By surveying adjacency connections of bounding boxes in mathematical formulae, we find that it is sufficient to deal with only graphs with small treewidth. We introduce a class of DAGs that is recursively defined with some graph-rewriting rules so that it contains only graphs with a bounded treewidth and covers most graphs corresponding to mathematical formulae. For the class of DAGs, we see that the problem can be solved in linear-time in the number of vertices of a graph.

This paper is organized as follows: In Section 2, we introduce a class of DAGs that is recursively defined with some graph-rewriting rules; in Section 3, a variation of the minimum spanning tree problem that is applicable to mathematical OCR is introduced and it is shown that the problem is NP-hard; in Section 4, we see how to solve the problem by graph reductions; in Section 5, the coverage of mathematical formulae with the class of DAGs and solutions for graphs not in the class is discussed; and in Section 6, the conclusion is drawn.

2. Recursively Defined DAGs

In this section, we introduce a class of DAGs that is recursively defined with some graph-rewriting rules.

First, we give some preliminary definitions. A *directed graph* is an ordered pair $G = (V, E)$, where V is a finite set, called *vertices*, and E is a finite set of ordered pairs of distinct vertices, called *edges*. An edge $e = (u, v)$ is called an *outgoing edge* of u and also called an *incoming edge* of v. A directed graph is a *directed acyclic graph (DAG)* if it has no directed cycles. For a DAG, a *source* is a vertex with no incoming edges, while a *sink* is a vertex with no outgoing edges. A DAG is *single-source* if it has exactly one source. Likewise, a DAG is *single-sink* if it has exactly one sink. A single-source DAG is a *rooted tree* if every vertex except the source has exactly one incoming edge. The source of a tree is also called the *root*, while sinks of a tree are also called *leaves*. A *spanning tree* of a DAG $G = (V, E)$ is a rooted tree $T = (V', E')$ such that $V' = V$ and $E' \subseteq E$.

Definition 1. The class *RD* is the smallest set of DAGs satisfying the following construction rules (See also Figure 3):

1. A DAG consisiting of a single vertex and no edges is in *RD*.
2. If $G = (V, E) \in RD$, $v \in V$ and $v' \notin V$, then the DAG $G' = (V \cup \{v'\}, E \cup \{(v, v')\})$ is in *RD*.
3. If $G = (V, E) \in RD$, $(u, v) \in E$ and $v' \notin V$, then the DAG $G' = (V \cup \{v'\}, (E - \{(u, v)\}) \cup \{(u, v'), (v', v)\})$ is in *RD*.
4. If $G = (V, E) \in RD$, $(u, v) \in E$ and $v' \notin V$, then the DAG $G' = (V \cup \{v'\}, E \cup \{(u, v'), (v', v)\})$ is in *RD*.
5. If $G = (V, E) \in RD$, $(u, v) \in E$ and $v_1, v_2 \notin V$, then the DAG $G' = (V \cup \{v_1, v_2\}, (E - \{(u, v)\}) \cup \{(u, v_1), (u, v_2), (v_1, v_2), (v_1, v), (v_2, v)\})$ is in *RD*.
6. If $G = (V, E) \in RD$, $(u, v) \in E$ and $v_1, v_2 \notin V$, then the DAG $G' = (V \cup \{v_1, v_2\}, E \cup \{(u, v_1), (u, v_2), (v_1, v_2), (v_1, v), (v_2, v)\})$ is in *RD*.

The class *RD* contains DAGs of treewidth at most 3. If the list of construction rules consists of only 1 and 2, then

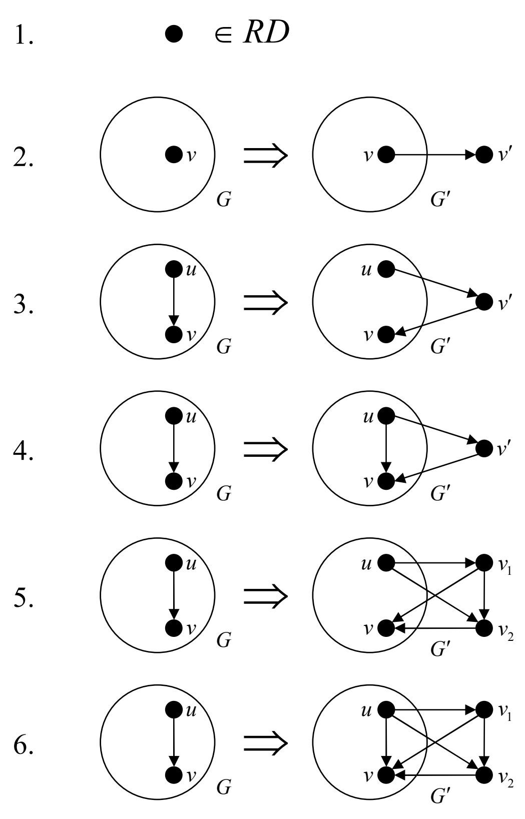


Figure 3: Construction rules for the class *RD*.

the class *RD* coincides with the set of all rooted trees. The construction rules from 1 to 4 yield only DAGs of treewidth at most 2.

Example 1. A DAG in the class RD and how it is constructed from a single-vertex graph are illustrated in Figure 4.

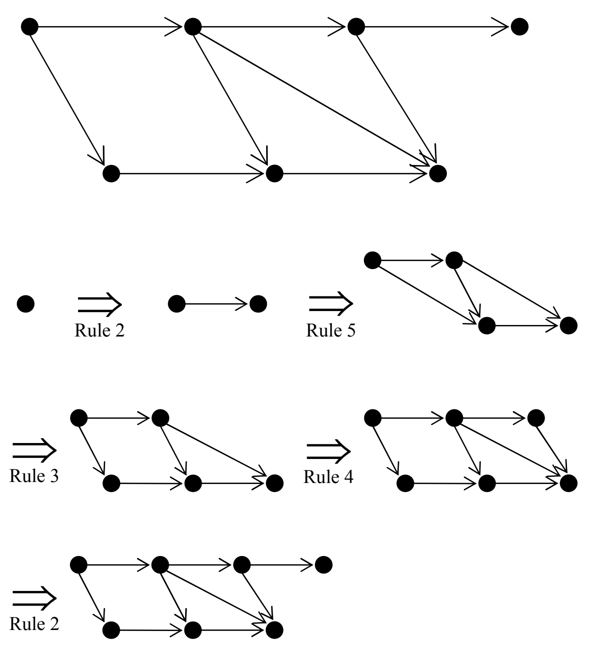


Figure 4: A DAG in the class RD and a derivation of it.

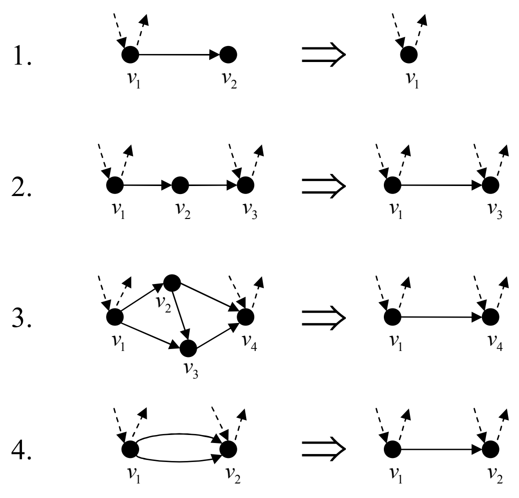


Figure 5: Reduction rules for the class *RD*.

By reversing the construction rules, we can obtain the following reduction rules. The reduction rules are useful for parsing graphs.

Proposition 1. *Any DAG in the class RD can be reduced to a single-vertex graph with the following reduction rules (See also Figure 5):*

1. *If the graph has an vertex v_2 with exactly one incoming edge (v_1, v_2) and no outgoing edges, then remove the vertex v_2 and the edge (v_1, v_2).*
2. *If the graph has an vertex v_2 with exactly one incoming edge (v_1, v_2) and one outgoing edge (v_2, v_3), then remove the vertex v_2 and the edges $(v_1, v_2), (v_2, v_3)$, and add an edge (v_1, v_3).*
3. *If the graph has two vertices v_2, v_3 such that v_2 has exactly one incoming edge (v_1, v_2) and two outgoing edges $(v_2, v_3), (v_2, v_4)$, and v_3 has exactly one incoming edge (v_1, v_3) and one outgoing edge (v_3, v_4), then remove the vertices v_2, v_3 and the edges $(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_2, v_4), (v_3, v_4)$, and add an edge (v_1, v_4).*
4. *If the graph has multiple edges (caused by the applications of the above rule 2 or 3), then replace them with a single edge.*

Example 2. How a DAG in the class RD can be reduced to a single-vertex graph is illustrated in Figure 6.

3. The Minimum Spanning Tree Problem

In this section, we introduce a variation of the minimum spanning tree problem that is applicable to mathematical OCR. Since the variation is rather complicated, we obtain the variation by extending the original problem step by step.

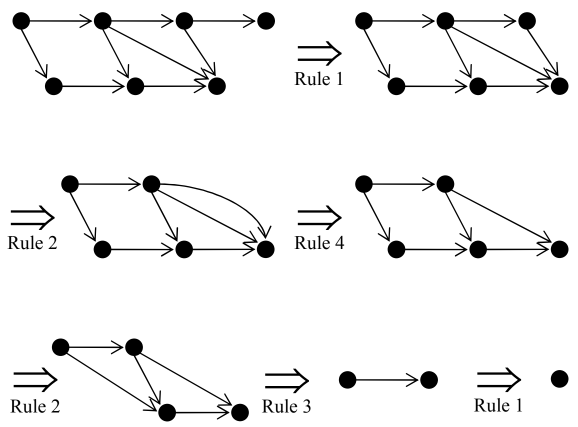


Figure 6: A reduction of a DAG in the class RD.

3.1. Variations of the Minimum Spanning Tree Problem

First, we introduce the original minimum spanning tree problem (MSTP). Let $\mathbb{R}$ be non-negative real numbers. MSTP is formulated as follows:

MSTP

Input: a connected, undirected graph $G = (V, E)$ and an assignment of weights to the edges $w : E \to \mathbb{R}$.

Output: a spanning tree $T = (V, E')$ of G such that $\sum_{e \in E'} w(e)$ is minimum.

Polynomial-time algorithms to solve MSTP are well-known such as Kruskal's algorithm [7] and Prim's algorithm [10]. These algorithms are commonly introduced in textbooks on algorithms.

Example 3. A connected, undirected graph with weighted edges and its minimum spanning tree are illustrated in Figure 7.

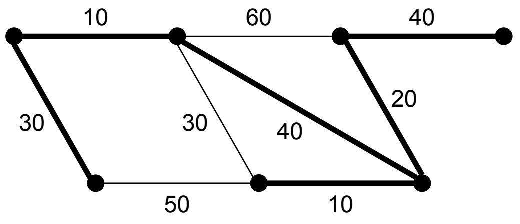


Figure 7: A problem instance of MSTP.

Though the original problem is defined on undirected graphs, we want to concentrate on problems defined on directed acyclic graphs (DAGs) in this paper. Since a DAG must be single-source to have a spanning tree, we will deal with only single-source DAGs.

MSTP for DAGs

Input: a single-source DAG $G = (V, E)$ and an assignment of weights to the edges $w : E \to \mathbb{R}$.

Output: a spanning tree $T = (V, E')$ of G such that $\sum_{e \in E'} w(e)$ is minimum.

MSTP for DAGs is much easier than the original problem because a minimum spanning tree can be obtained by choosing an incoming edge with the minimum weight for each vertex except the source.

Example 4. A single-source DAG with weighted edges and its minimum spanning tree are illustrated in Figure 8.

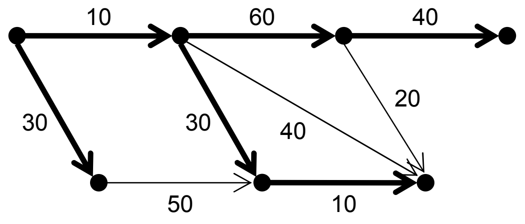


Figure 8: A problem instance of MSTP for DAGs.

In order to obtain a variation of MSTP for the application to mathematical OCR, we need to import the notion of "candidate selection" into MSTP. For the sake of simplicity, we first introduce a decision problem to determine whether or not there exists a proper selection of candidates on a spanning tree of a given graph. We call this problem "the spanning tree problem with candidate selection."

STP with Candidate Selection

Input: a single-source DAG $G = (V, E)$ such that each vertex $v \in V$ has a non-empty set of candidates C_v and each edge $(v_1, v_2) \in E$ has a non-empty set of links $L_{(v_1,v_2)} \subseteq C_{v_1} \times C_{v_2}$.

Output: "*yes*" if there exist a selection of candidate $s_v \in C_v$ for each $v \in V$ and a spanning tree $T = (V, E')$ of G such that the link (s_{v_1}, s_{v_2}) is in $L_{(v_1,v_2)}$ for each $(v_1, v_2) \in E'$, or "*no*" otherwise.

Unfortunately, it will be shown that this problem is NP-complete.

Example 5. A single-source DAG with candidates and one of its spanning trees are illustrated in Figure 9. Vertices are indicated by dotted rectangles, candidates are represented by circled symbols, and links are drawn between candidates. Edges are not illustrated because we know that there is an edge where a link exists. The shape of the DAG is same as the DAG in Example 4. There exist many other proper selections of candidates and spanning trees.

Putting weights on the links, we introduce a minimization version of the decision problem.

MSTP with Candidate Selection

Input: a single-source DAG $G = (V, E)$ such that each vertex $v \in V$ has a non-empty set of candidates C_v and each edge $(v_1, v_2) \in E$ has a non-empty set of links $L_{(v_1,v_2)} \subseteq C_{v_1} \times C_{v_2}$, and an assignment of weights to

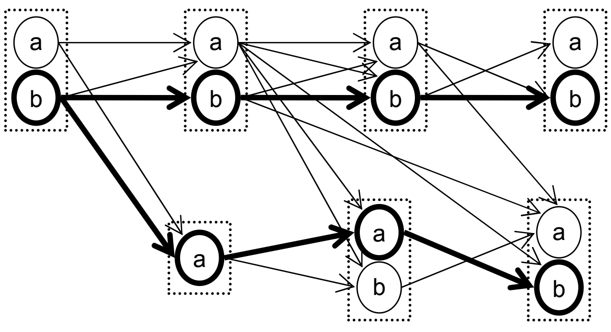


Figure 9: A problem instance of STP with candidate selection.

the links such that if $e \in E$ and $(c_1, c_2) \in L_e$, then $w(e, c_1, c_2) \in \mathbb{R}$, otherwise $w(e, c_1, c_2)$ is undefined.

Output: a selection of candidate $s_v \in C_v$ for each $v \in V$ and a spanning tree $T = (V, E')$ of G such that the link (s_{v_1}, s_{v_2}) is in $L_{(v_1,v_2)}$ for each $(v_1, v_2) \in E'$ and $\sum_{(v_1,v_2)\in E'} w((v_1, v_2), s_{v_1}, s_{v_2})$ is minimum.

Since STP with candidates selection is NP-complete, this problem is NP-hard.

Example 6. A single-source DAG with candidates and weighted links and its minimum spanning tree are illustrated in Figure 10.

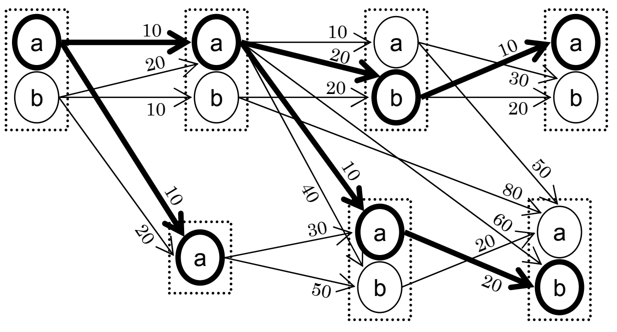


Figure 10: A problem instance of MSTP with candidate selection.

At last, we are ready to introduce the variation of MSTP that is applicable to mathematical OCR. In this variation, all the links are labeled, and we have to select both candidates and link-labels so that there exists a spanning tree such that all outgoing links have distinct labels for each candidate.

MSTP with Candidate and Link-Label Selections

Input: a set of link-labels $\mathbb{L}$, a single-source DAG $G = (V, E)$ such that each vertex $v \in V$ has a non-empty set of candidates C_v and each edge $(v_1, v_2) \in E$ has a non-empty set of links $L_{(v_1,v_2)} \subseteq C_{v_1} \times C_{v_2} \times \mathbb{L}$, and an assignment of weights to the links such that if $e \in E$ and $(c_1, c_2, l) \in L_e$, then $w(e, c_1, c_2, l) \in \mathbb{R}$, otherwise $w(e, c_1, c_2, l)$ is undefined.

Output: a selection of candidate $s_v \in C_v$ for each $v \in V$, a spanning tree $T = (V, E')$ of G, and a selection of link-label $l_e \in \mathbb{L}$ for each $e \in E'$ such that the link $(s_{v_1}, s_{v_2}, l_{(v_1,v_2)})$ is in $L_{(v_1,v_2)}$ for each $(v_1, v_2) \in E'$, $\sum_{(v_1,v_2)\in E'} w((v_1, v_2), s_{v_1}, s_{v_2}, l_{(v_1,v_2)})$ is minimum, and, for each $v_1 \in V$, if $(v_1, v_2), (v_1, v_3) \in E'$ and $v_2 \neq v_3$, then $l_{(v_1,v_2)} \neq l_{(v_1,v_3)}$.

Similarly, this problem is also NP-hard.

Example 7. A single-source DAG with candidates and labeled, weighted links (corresponding to the virtual link network in Figure 2) and its minimum spanning tree are illustrated in Figure 11. There exist multiple links between candidates if each of them has a different link-label.

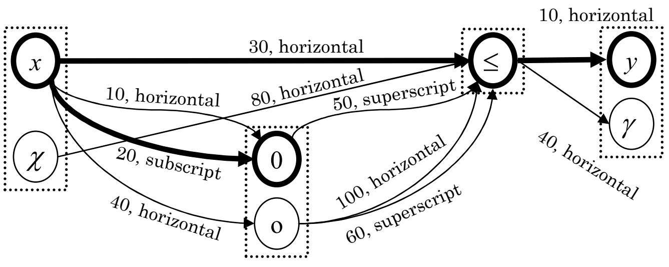


Figure 11: A problem instance of MSTP with candidate and link-label selections.

3.2. NP-Completeness of the Spanning Tree Problem with Candidate Selection

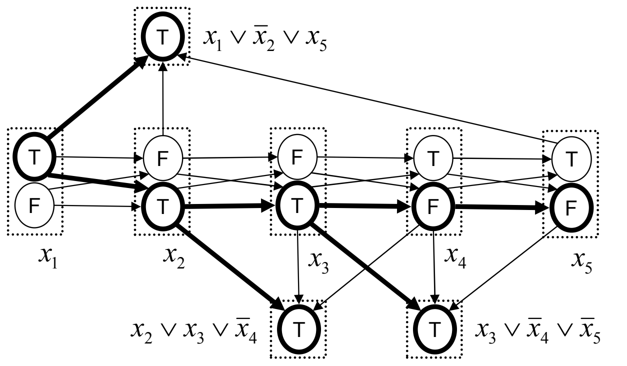
We will prove that the spanning tree problem with candidate selection is NP-complete by reducing the Boolean satisfiability problem (SAT) to this problem. SAT is the problem of determining if there exists an assignment of Boolean values to the variables of a given Boolean formula in conjunctive normal form so that it makes the formula true. SAT is one of the most famous NP-complete problems.

Theorem 1. *The spanning tree problem with candidate selection is NP-complete.*

Proof. We will show the NP-hardness by reducing the Boolean satisfiability problem (SAT) to this problem.

Let $\mathcal{F}$ be a given Boolean formula in conjunctive normal form, where $\mathcal{C} = \{c_1, \dots, c_m\}$ is the set of clauses composing $\mathcal{F}$, and $\mathcal{X} = \{x_1, \dots, x_n\}$ is the set of Boolean variables appearing in $\mathcal{F}$.

From $\mathcal{F}$, we construct a DAG with candidates and links $G = (V, E)$ as follows: $V = \{x_1, \dots, x_n\} \cup \{c_1, \dots, c_m\}$. For $v \in \{x_1, \dots, x_n\}$, $C_v = \{T, F\}$, and, for $v \in \{c_1, \dots, c_m\}$, $C_v = \{T\}$. $E = E_1 \cup E_2$, where $E_1 = \{(x_i, x_{i+1}) \mid 1 \leq i \leq n-1\}$ and $E_2 = \{(x_i, c_j) \mid x_i$ or $\bar{x}_i$ appears in c_j for $1 \leq i \leq n$ and $1 \leq j \leq m\}$. For $e \in E_1$, $L_e = \{(T,T), (T,F), (F,T), (F,F)\}$, and, for $(x_i, c_j) \in E_2$, $L_{(x_i,c_j)} = \{(T,T)\}$ if x_i appears in c_j, or $L_{(x_i,c_j)} = \{(F,T)\}$ if $\bar{x}_i$ appears in c_j.



$x_1 \vee \bar{x}_2 \vee x_5$
T
F
x_1
x_2
x_3
x_4
x_5
$x_2 \vee x_3 \vee \bar{x}_4$
$x_3 \vee \bar{x}_4 \vee \bar{x}_5$

Figure 12: The graph corresponding to the formula, and a proper selection of candidates and one of spanning trees of it.

The construction of the DAG G can be done in polynomial time.

For example, the DAG corresponding to the formula $(x_1 \vee \bar{x}_2 \vee x_5) \wedge (x_2 \vee x_3 \vee \bar{x}_4) \wedge (x_3 \vee \bar{x}_4 \vee \bar{x}_5)$ is illustrated in Figure 12. A proper selection of candidates and one of spanning trees of the DAG is also illustrated.

We will prove the following statement: *$\mathcal{F}$ has a truth assignment if and only if there exists a proper selection of candidates and a spanning tree of G.*

The 'only if' part is proved as follows. Suppose that $(x_1, \dots, x_n) = (a_1, \dots, a_n)$ is a truth assignment of $\mathcal{F}$, where $a_1, \dots, a_n \in \{T, F\}$. Then, for each c_j, there is at least one variable in c_j which makes the clause true. Let $x_{c_1}, \dots, x_{c_m}$ be such variables. If we set $(s_{x_1}, \dots, s_{x_n}, s_{c_1}, \dots, s_{c_m}) = (a_1, \dots, a_n, T, \dots, T)$ and $E' = E_1 \cup \{\{x_{c_j}, c_j\} \mid 1 \le j \le m\}$, then $(s_{x_1}, \dots, s_{x_n}, s_{c_1}, \dots, s_{c_m})$ is a proper selection of candidates and $T = (V, E')$ is a spanning tree of G.

The 'if' part is proved as follows. Suppose that $(s_{x_1}, \dots, s_{x_n}, s_{c_1}, \dots, s_{c_m})$ is a proper selection of candidates and $T = (V, E')$ is a spanning tree of G. Then, $(x_1, \dots, x_n) = (s_{x_1}, \dots, s_{x_n})$ is a truth assignment of $\mathcal{F}$.

Since we can reduce SAT to this problem, the NP-hardness of the problem is proved.

On the other hand, given a DAG with candidates and links G, we can nondeterministically obtain a selection of candidates and a spanning tree T and check if T is proper in polynomial time. This means that the problem is in the class NP.

Therefore, the problem is NP-complete. □

Since MSTP with candidate selection and MSTP with candidate and link-label selections are extensions of the spanning tree problem with candidate selection, the following corollaries are obtained.

Corollary 1. *The minimum spanning tree problem with candidate selection is NP-hard.*

Corollary 2. *The minimum spanning tree problem with candidate and link-label selections is NP-hard.*

4. Solving the Minimum Spanning Tree Problem by Graph Reductions

In this section, we see that, for DAGs in the class RD, all the variations of MSTP introduced in the previous section can be solved in linear-time in the number of vertices of a graph. We take advantage of graph reductions using the reduction rules of the class RD.

4.1. Solution to MSTP for DAGs

We first explain how to solve MSTP for DAGs. The solution is based on graph reductions using the reduction rules of the class RD. In order to describe the solution, we need to allow vertices and edges to hold additional information because remaining vertices and edges have to maintain essential information concerning removed vertices and edges during graph reductions. Thus, we will consider the following generalization of MSTP, where each vertex has a weight and each edge has two kinds of weights.

MSTP for Weighted Vertices and Bi-Weighted Edges

Input: a single-source DAG $G = (V, E)$, an assignment of weights to the vertices $w' : V \to \mathbb{R}$, and two assignments of weights to the edges $w : E \to \mathbb{R}$, $\overline{w} : E \to \mathbb{R}$.

Output: a spanning tree $T = (V, E')$ of G such that $\sum_{v \in V} w'(v) + \sum_{e \in E'} w(e) + \sum_{e \in E - E'} \overline{w}(e)$ is minimum.

For $e \in E$, we call $\overline{w}(e)$ the *hidden weight* of e. Obviously, if we set $w'(v) = 0$ for all $v \in V$ and $\overline{w}(e) = 0$ for all $e \in E$, then the problem is same as the original problem.

As we know, any graph in the class RD can be reduced to a single-vertex graph. When a graph is finally reduced to a single-vertex graph, the problem is solved if the weight of the sole vertex is the same as the minimum spanning trees of the original graph. We will see how to change the weights of vertices and edges so that the weight of minimum spanning trees is unchanged with graph reductions.

Every time when a graph is reduced, we change the weights of vertices and edges as follows:

1. If the graph has an vertex v_2 with exactly one incoming edge (v_1, v_2) and no outgoing edges, then remove the vertex v_2 and the edge (v_1, v_2), and add $w'(v_2) + w((v_1, v_2))$ to $w'(v_1)$.
2. If the graph has an vertex v_2 with exactly one incoming edge (v_1, v_2) and one outgoing edge (v_2, v_3), then remove the vertex v_2 and the edges $(v_1, v_2), (v_2, v_3)$, add a new edge e from v_1 to v_3, and set $w(e) = w'(v_2) + w((v_1, v_2)) + w((v_2, v_3))$ and $\overline{w}(e) = w'(v_2) + w((v_1, v_2)) + \overline{w}((v_2, v_3))$.
3. If the graph has two vertices v_2, v_3 such that v_2 has exactly one incoming edge (v_1, v_2) and two outgoing edges $(v_2, v_3), (v_2, v_4)$, and v_3 has exactly one incoming edge (v_1, v_3) and one outgoing edge (v_3, v_4), then remove the vertices v_2, v_3 and the edges

$(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_2, v_4), (v_3, v_4)$, add a new edge e from v_1 to v_4, and set

$$\begin{aligned}w(e) = \min(&w((v_1,v_2)) + w((v_1,v_3)) + \overline{w}((v_2,v_3)) + \\ &w((v_2,v_4)) + \overline{w}((v_3,v_4)) + w'(v_2) + w'(v_3), \\ &w((v_1,v_2)) + w((v_1,v_3)) + \overline{w}((v_2,v_3)) + \\ &\overline{w}((v_2,v_4)) + w((v_3,v_4)) + w'(v_2) + w'(v_3), \\ &w((v_1,v_2)) + \overline{w}((v_1,v_3)) + w((v_2,v_3)) + \\ &w((v_2,v_4)) + \overline{w}((v_3,v_4)) + w'(v_2) + w'(v_3), \\ &w((v_1,v_2)) + \overline{w}((v_1,v_3)) + w((v_2,v_3)) + \\ &\overline{w}((v_2,v_4)) + w((v_3,v_4)) + w'(v_2) + w'(v_3))\end{aligned}$$

and

$$\begin{aligned}\overline{w}(e) = \min(&w((v_1,v_2)) + w((v_1,v_3)) + \overline{w}((v_2,v_3)) + \\ &\overline{w}((v_2,v_4)) + \overline{w}((v_3,v_4)) + w'(v_2) + w'(v_3), \\ &w((v_1,v_2)) + \overline{w}((v_1,v_3)) + w((v_2,v_3)) + \\ &\overline{w}((v_2,v_4)) + \overline{w}((v_3,v_4)) + w'(v_2) + w'(v_3)).\end{aligned}$$

4. If the graph has multiple edges e, e' from v_1 to v_2 (caused by the applications of the reduction rule 2 or 3), then replace them with a new edge e'', and set $w(e'') = \min(w(e) + \overline{w}(e'), \overline{w}(e) + w(e'))$ and $\overline{w}(e'') = \overline{w}(e) + \overline{w}(e')$.

In the cases 3 and 4, plural edges are reduced to an edge. At this time, some weights of edges are selected, and the other weights of edges are abandoned. If we remember which weights of edges are surviving, we can obtain a minimum spanning tree of the original graph.

Example 8. How the weights of the vertices and edges of a DAG are changed with graph reductions is illustrated in Figure 13. The problem instance used here is the same as in Example 4.

For the first configuration (1), we set all the weights of vertices to be 0 and set all the hidden weights of edges to be 0. The weights of vertices and the hidden weights of edges are described with round brackets and square brackets, respectively.

(1) ⇒ (2) : The upper-rightmost vertex and its incoming edge are removed. The sum of the weights of them, $0+40 = 40$, is added to the weight of the parent vertex.

(2) ⇒ (3) : The upper-rightmost vertex and its incoming and outgoing edges are replaced with a new edge. The sum of the weights of them, $40 + 60 + 20 = 120$, becomes the weight of the new edge. The sum of the weights of the vertex and the incoming edge and the hidden weight of the outgoing edge, $40 + 60 + 0 = 100$, becomes the hidden weight of the new edge.

(3) ⇒ (4) : The multiple edges are replaced with a new edge. The smaller sum of the weights of an edge and the hidden weight of another edge, $\min(120+0, 40+100)=120$, becomes the weight of the new edge. The sum of the hidden weights of the edges, $100 + 0 = 100$, becomes the hidden weight of the new edge.

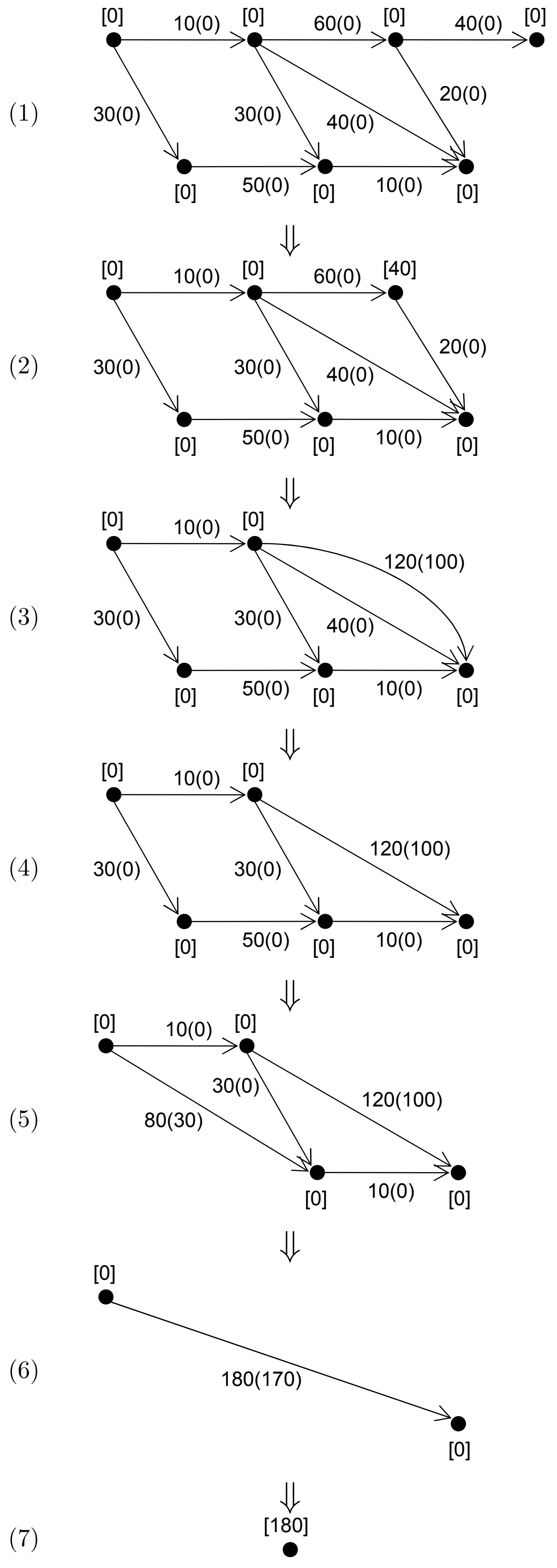


Figure 13: How the weights of the vertices and edges of a DAG are changed with graph reductions.

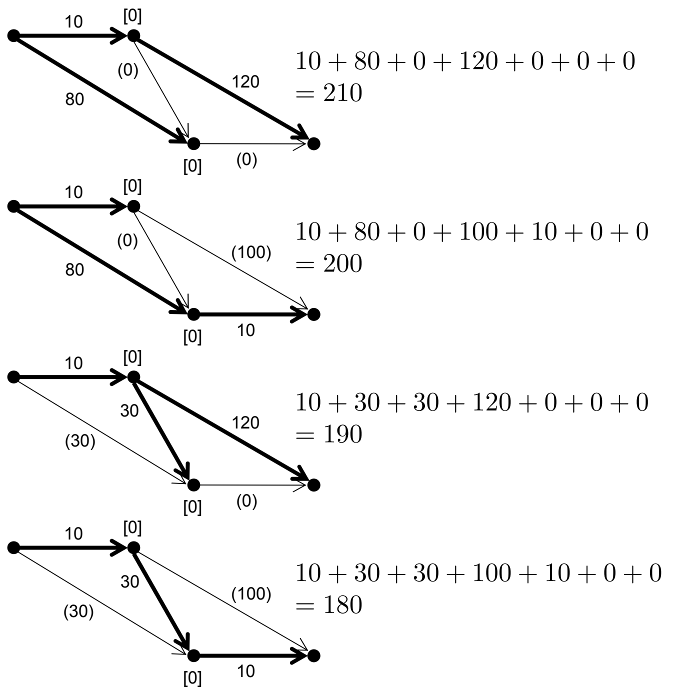


Figure 14: The trees spanning the four vertices.

(4) ⇒ (5) : The lower-leftmost vertex and its incoming and outgoing edges are replaced with a new edge. The sum of the weights of them, $0+30+50 = 80$, becomes the weight of the new edge. The sum of the weights of the vertex and the incoming edge and the hidden weight of the outgoing edge, $0 + 30 + 0 = 30$, becomes the hidden weight of the new edge.

(5) ⇒ (6) : The two vertices at the middle and all the edges connected with them are replaced with a new edge. As illustrated in Figure 14, the weights of trees spanning the four vertices are examined. The minimum weight among the four trees is $\min(210, 200, 190, 180) = 180$. The sum of the weights of the two vertices at the middle and the minimum weight of trees spanning the four vertices, 180, becomes the weight of the new edge. On the other hand, as illustrated in Figure 15, the weights of trees spanning the three vertices are examined. The minimum weight among the two trees is $\min(190, 170) = 170$. The sum of the weights of the two vertices at the middle and the minimum weight of trees spanning the three vertices, 170, becomes the hidden weight of the new edge.

(6) ⇒ (7) : The upper-right vertex and its incoming edge are removed. The sum of the weights of them, $0 + 180 = 180$, is added to the weight of the parent vertex.

Concerning (7), the remaining vertex has the same weight as the weight of minimum spanning trees of the original graph.

4.2. Solution to MSTP with candidate selection

We solve MSTP with candidate selection in a similar way; we assume that each candidate has a weight and each link

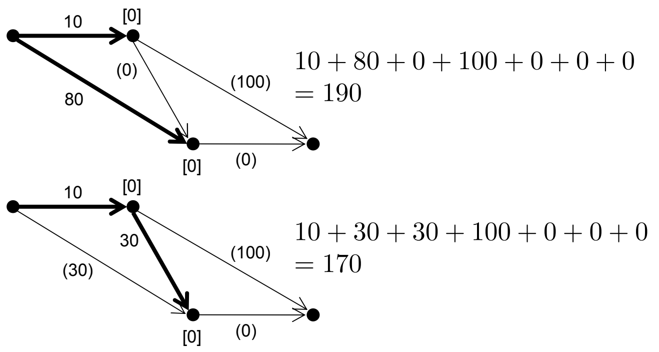


Figure 15: The trees spanning the three vertices.

has two kinds of weights, and every time when a graph is reduced, we change the weights of candidates and links so that the reduced graph has minimum spanning trees of the same weight.

In order to describe the solution, we will consider the following generalization of MSTP with candidate selection.

MSTP with Candidate Selection for Weighted Vertices and Bi-Weighted Edges

Input: a single-source DAG $G = (V, E)$ such that each vertex $v \in V$ has a non-empty set of candidates C_v and each edge $(v_1, v_2) \in E$ has the full set of links between candidates $L_{(v_1,v_2)} = C_{v_1} \times C_{v_2}$, an assignment of weights to the candidates such that, for each $v \in V$ and $c \in C_v$, $w'(v, c) \in \mathbb{R}$, and two assignments of weights to the links such that, for each $e \in E$ and $(c_1, c_2) \in L_e$, $w(e, c_1, c_2) \in \mathbb{R}$ and $\overline{w}(e, c_1, c_2) \in \mathbb{R}$.

Output: a selection of candidate $s_v \in C_v$ for each $v \in V$ and a spanning tree $T = (V, E')$ of G such that $\sum_{v \in V} w'(v, s_v) + \sum_{(v_1,v_2) \in E'} w((v_1, v_2), s_{v_1}, s_{v_2}) + \sum_{(v_1,v_2) \in E-E'} \overline{w}((v_1, v_2), s_{v_1}, s_{v_2})$ is minimum.

For $e \in E$ and $(c_1, c_2) \in L_e$, we call $\overline{w}(e, c_1, c_2)$ the *hidden weight* of the link (c_1, c_2) of e. Note that $w(e, c_1, c_2)$ and $\overline{w}(e, c_1, c_2)$ can be infinity ∞ for some $e \in E$ and $(c_1, c_2) \in L_e$. Given a problem instance of the original problem, if we set $w'(v, c) = 0$ for all $v \in V$ and $c \in C_v$, $\overline{w}(e, c_1, c_2) = 0$ for all $e \in E$ and $(c_1, c_2) \in L_e$, and $w(e, c_1, c_2) = \infty$ for all $e \in E$ and $(c_1, c_2) \notin L_e$ of the original problem instance, then the problem is almost same as the original problem except that each edge has the full set of links between candidates. In the generalized problem, a minimum spanning tree always exists. If no minimum spanning tree exists in the original problem instance, then the weight of minimum spanning trees becomes infinity.

Every time when a graph is reduced, we change the weights of vertices and edges as follows:

1. In case the graph has an vertex v_2 with exactly one incoming edge (v_1, v_2) and no outgoing edges: Remove the vertex v_2 and the edge (v_1, v_2); and, for each $c_1 \in C_{v_1}$, add $\min_{c_2 \in C_{v_2}}(w'(v_2, c_2) + w((v_1, v_2), c_1, c_2))$ to $w'(v_1, c_1)$.

2. In case the graph has an vertex v_2 with exactly one incoming edge (v_1, v_2) and one outgoing edge (v_2, v_3): Remove the vertex v_2 and the edges $(v_1, v_2), (v_2, v_3)$; add a new edge e from v_1 to v_3; and, for each $c_1 \in C_{v_1}$ and $c_3 \in C_{v_3}$, set

$$w(e, c_1, c_3) = \min_{c_2 \in C_{v_2}} (w'(v_2, c_2) + w((v_1, v_2), c_1, c_2) + w((v_2, v_3), c_2, c_3))$$

and

$$\overline{w}(e, c_1, c_3) = \min_{c_2 \in C_{v_2}} (w'(v_2, c_2) + w((v_1, v_2), c_1, c_2) + \overline{w}((v_2, v_3), c_2, c_3)).$$

3. In case the graph has two vertices v_2, v_3 such that v_2 has exactly one incoming edge (v_1, v_2) and two outgoing edges $(v_2, v_3), (v_2, v_4)$, and v_3 has exactly one incoming edge (v_1, v_3) and one outgoing edge (v_3, v_4): Remove the vertices v_2, v_3 and the edges $(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_2, v_4), (v_3, v_4)$; add a new edge e from v_1 to v_4; and, for each $c_1 \in C_{v_1}$ and $c_4 \in C_{v_4}$, set $w(e, c_1, c_4) = \min(w_1(c_1, c_4), w_2(c_1, c_4), w_3(c_1, c_4), w_4(c_1, c_4))$ and $\overline{w}(e, c_1, c_2) = \min(\overline{w_1}(c_1, c_4), \overline{w_2}(c_1, c_4))$ where:

$$\begin{aligned}
w_1(c_1, c_4) = \min_{c_2 \in C_{v_2}, c_3 \in C_{v_3}} & (w'(v_2, c_2) + w'(v_3, c_3) + \\
& w((v_1, v_2), c_1, c_2) + w((v_1, v_3), c_1, c_3) + \\
& \overline{w}((v_2, v_3), c_2, c_3) + w((v_2, v_4), c_2, c_4) + \\
& \overline{w}((v_3, v_4), c_3, c_4)), \\
w_2(c_1, c_4) = \min_{c_2 \in C_{v_2}, c_3 \in C_{v_3}} & (w'(v_2, c_2) + w'(v_3, c_3) + \\
& w((v_1, v_2), c_1, c_2) + w((v_1, v_3), c_1, c_3) + \\
& \overline{w}((v_2, v_3), c_2, c_3) + \overline{w}((v_2, v_4), c_2, c_4) + \\
& w((v_3, v_4), c_3, c_4)), \\
w_3(c_1, c_4) = \min_{c_2 \in C_{v_2}, c_3 \in C_{v_3}} & (w'(v_2, c_2) + w'(v_3, c_3) + \\
& w((v_1, v_2), c_1, c_2) + \overline{w}((v_1, v_3), c_1, c_3) + \\
& w((v_2, v_3), c_2, c_3) + w((v_2, v_4), c_2, c_4) + \\
& \overline{w}((v_3, v_4), c_3, c_4)), \\
w_4(c_1, c_4) = \min_{c_2 \in C_{v_2}, c_3 \in C_{v_3}} & (w'(v_2, c_2) + w'(v_3, c_3) + \\
& w((v_1, v_2), c_1, c_2) + \overline{w}((v_1, v_3), c_1, c_3) + \\
& w((v_2, v_3), c_2, c_3) + \overline{w}((v_2, v_4), c_2, c_4) + \\
& w((v_3, v_4), c_3, c_4))
\end{aligned}$$

and

$$\begin{aligned}
\overline{w_1}(c_1, c_4) = \min_{c_2 \in C_{v_2}, c_3 \in C_{v_3}} & (w'(v_2, c_2) + w'(v_3, c_3) + \\
& w((v_1, v_2), c_1, c_2) + w((v_1, v_3), c_1, c_3) + \\
& \overline{w}((v_2, v_3), c_2, c_3) + \overline{w}((v_2, v_4), c_2, c_4) + \\
& \overline{w}((v_3, v_4), c_3, c_4)), \\
\overline{w_2}(c_1, c_4) = \min_{c_2 \in C_{v_2}, c_3 \in C_{v_3}} & (w'(v_2, c_2) + w'(v_3, c_3) + \\
& w((v_1, v_2), c_1, c_2) + \overline{w}((v_1, v_3), c_1, c_3) + \\
& w((v_2, v_3), c_2, c_3) + \overline{w}((v_2, v_4), c_2, c_4) + \\
& \overline{w}((v_3, v_4), c_3, c_4)).
\end{aligned}$$

4. In case the graph has multiple edges e, e' from v_1 to v_2 (caused by the applications of the reduction rule 2 or 3): Replace them with a new edge e''; and, for each $c_1 \in C_{v_1}$ and $c_2 \in C_{v_2}$, set

$$w(e'', c_1, c_2) = \min(w(e, c_1, c_2) + \overline{w}(e', c_1, c_2), \overline{w}(e, c_1, c_2) + w(e', c_1, c_2))$$

and

$$\overline{w}(e'', c_1, c_2) = \overline{w}(e, c_1, c_2) + \overline{w}(e', c_1, c_2).$$

In all the cases, some weights of links and candidates are selected, and the other weights are abandoned. If we remember which weights of links and candidates are surviving, we can obtain a minimum spanning tree of the original graph with a proper selection of candidates.

Example 9. How the weights of the links and candidates of a DAG are changed with graph reductions is illustrated in Figure 16 and 17. The problem instance used here is the same as in Example 6.

For the first configuration (1), we set all the weights of candidates to be 0 and set all the hidden weights of links to be 0. The weights of candidates and the hidden weights of links are described with round brackets and square brackets, respectively. If a link does not exist between a pair of candidates, we suppose that there is a link with a weight of infinity and a hidden weight of 0.

(1) $\Rightarrow$ (2) : The upper-rightmost vertex and its incoming edge are removed. How the weights of candidates a, b of the parent vertex v are obtained is illustrated in Figure 18. The weights of candidates of the parent vertex v are the smallest sum of the weights of a link of the incoming edge and a candidate of the upper-rightmost vertex.

(2) $\Rightarrow$ (3) : The upper-rightmost vertex and its incoming and outgoing edges are replaced with a new edge e. How the weights and hidden weights of the links are obtained is illustrated in Figure 19. The weights of links of the new edge are the smallest sum of the weights of links of the replaced edges and a candidate of the replaced vertex.

(3) $\Rightarrow$ (4) : The multiple edges are replaced with a new edge e. The smaller sum of the weights of a link of an edge and the hidden weight of a link of another edge becomes the weight of links the new edge. The sum of the hidden weights of links of the edges becomes the hidden weight of links of the new edge.

(4) $\Rightarrow$ (5) : The lower-leftmost vertex and its incoming and outgoing edges are replaced with a new edge. The weights of links of the new edge are the smallest sum of the weights of links of the replaced edges and a candidate of the replaced vertex.

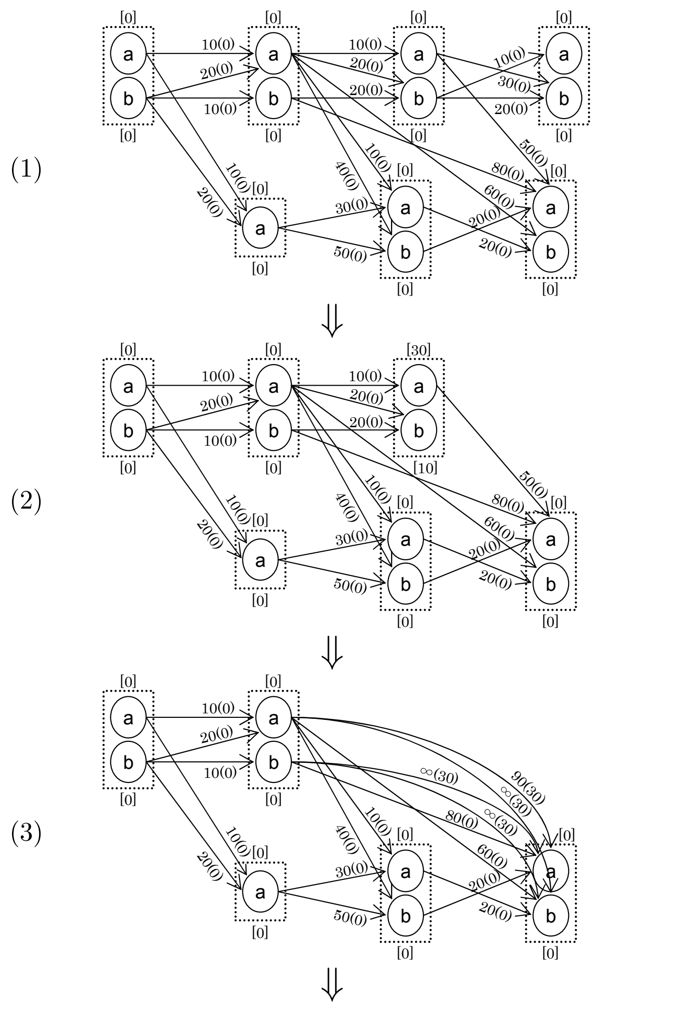


Figure 16: How the weights of the links and candidates of a DAG are changed with graph reductions (Part. 1/2).

(5) ⇒ (6) : The two vertices at the middle and all the edges connected with them are replaced with a new edge e. As illustrated in Figure 20, the weights of trees spanning the four vertices are examined. For each pair of candidates, the minimum weight among the four trees is calculated. They become the weights of links of the new edge. For the hidden weight of links, as illustrated in Figure 21, the weights of trees spanning the three vertices are examined. For each pair of candidates, the smaller weight among the two trees becomes the hidden weight of a link of the new edge.

(6) ⇒ (7) : The upper-right vertex and its incoming edge are removed. The smallest sum of the weights of a link of the incoming edge and a candidate of the upper-rightmost vertex, min(90+0, 110+0) and min(110+0, 100+0), become the weights of candidates of the parent vertex.

At last, the remaining vertex has candidates one of which has the same weight as the weight of minimum spanning trees of the original graph. The value 90 is the minimum weight of spanning trees if the candidate a is chosen for the root. Likewise, the value 100 is the minimum weight of spanning trees if the candidate b is chosen for the root.

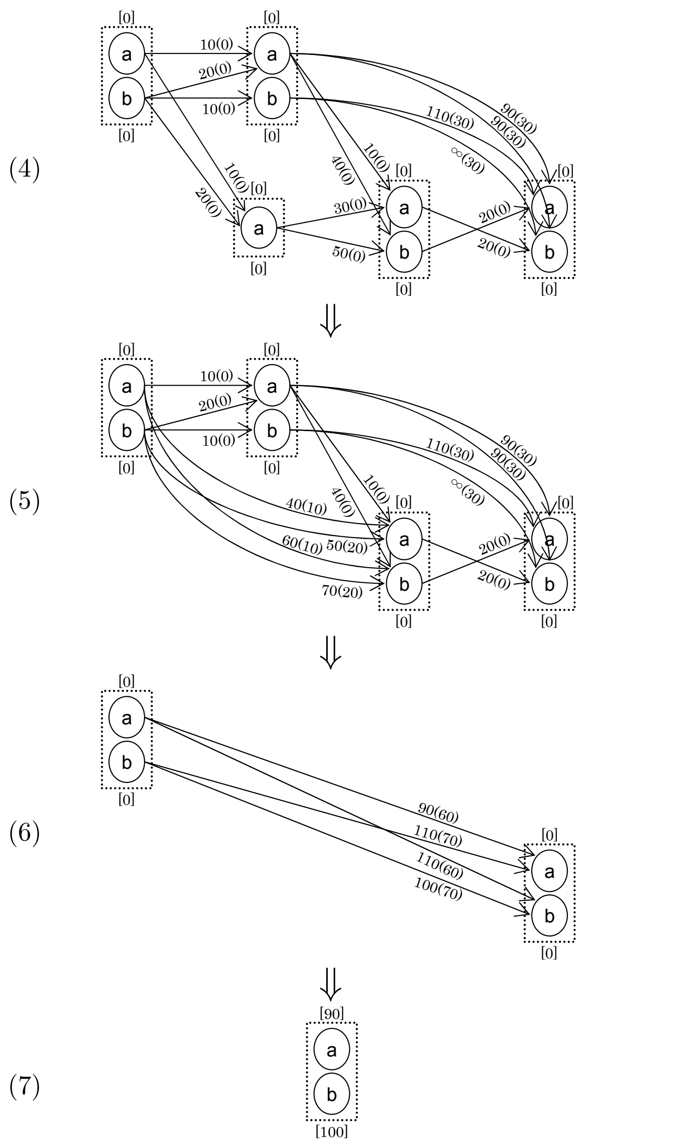
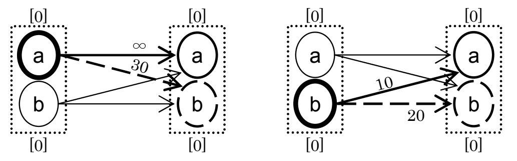


Figure 17: How the weights of the links and candidates of a DAG are changed with graph reductions (Part. 2/2).

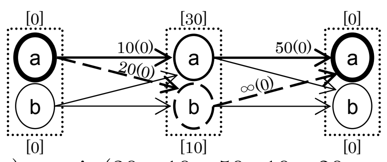


$$w'(v,a) = \min(0+\infty, 0+30) = 30$$
$$w'(v,b) = \min(0+10, 0+20) = 20$$

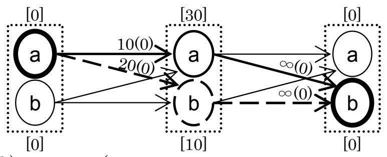
Figure 18: The weights of candidates of the parent vertex.

4.3. Solution to MSTP with candidate and link-label selections

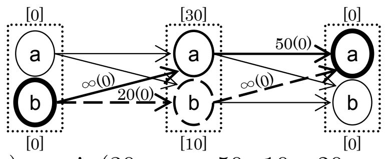
Now, we are ready to solve MSTP with candidate and link-label selections. Similarly, we assume that each candidate has a weight and each link has two kinds of weights. In addition to that, each link has a set of link-labels. Every time when a graph is reduced, we change the weights of



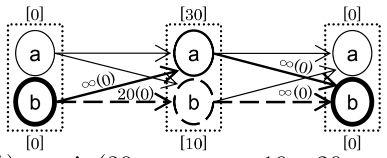
$w(e,a,a) = \min(30+10+50,\ 10+20+\infty) = 90$
$\overline{w}(e,a,a) = \min(30+10+0,\ 10+20+0) = 30$



$w(e,a,b) = \min(30+10+\infty,\ 10+20+\infty) = \infty$
$\overline{w}(e,a,b) = \min(30+10+0,\ 10+20+0) = 30$



$w(e,b,a) = \min(30+\infty+50,\ 10+20+\infty) = \infty$
$\overline{w}(e,b,a) = \min(30+\infty+0,\ 10+20+0) = 30$



$w(e,b,b) = \min(30+\infty+\infty,\ 10+20+\infty) = \infty$
$\overline{w}(e,b,b) = \min(30+\infty+0,\ 10+20+0) = 30$

Figure 19: The trees spanning the three vertices.

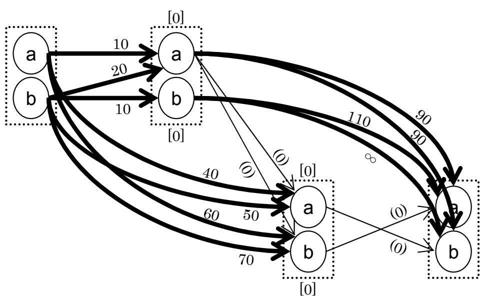
candidates and links and the sets of link-labels so that the reduced graph has minimum spanning trees of the same weight.

In order to describe the solution, we will consider the following generalization of MSTP with candidate and link-label selections.

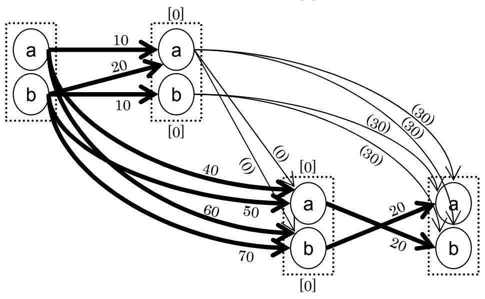
MSTP with Candidate and Link-Label Selections for Weighted Vertices and Bi-Weighted Edges

Input: a set of link-labels $\mathbb{L}$, a single-source DAG $G = (V, E)$ such that each vertex $v \in V$ has a non-empty set of candidates C_v and each edge $(v_1, v_2) \in E$ has a non-empty set of links $L_{(v_1,v_2)} \subseteq C_{v_1} \times C_{v_2} \times 2^{\mathbb{L}}$ ($2^{\mathbb{L}}$ is the power set of $\mathbb{L}$) such that $(c_1, c_2, \emptyset) \in L_{(v_1,v_2)}$ for each $(c_1, c_2) \in C_{v_1} \times C_{v_2}$, an assignment of weights to the candidates with a link-label set such that $w'(v, c, \mathcal{L}) \in \mathbb{R}$ for each $v \in V$, $c \in C_v$ and $\mathcal{L} \subseteq \mathbb{L}$, and two assignments of weights to the links such that if $e \in E$ and $(c_1, c_2, \mathcal{L}) \in L_e$, then $w(e, c_1, c_2, \mathcal{L}) \in \mathbb{R}$ and $\overline{w}(e, c_1, c_2, \mathcal{L}) \in \mathbb{R}$, otherwise $w(e, c_1, c_2, \mathcal{L})$ and $\overline{w}(e, c_1, c_2, \mathcal{L})$ are undefined.

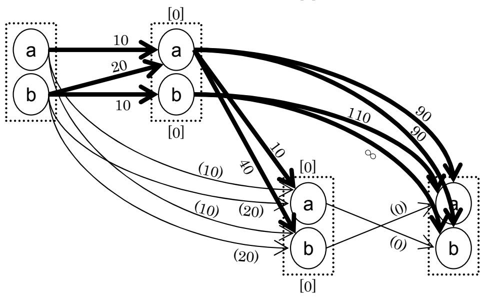
Output: a selection of candidate $s_v \in C_v$ for each $v \in V$, a spanning tree $T = (V, E')$ of G, a selection of link-label set $\mathcal{L}_x \subseteq \mathbb{L}$ for each $x \in V \cup E'$ such that, for each $v_1 \in V$, if $(v_1, v_2), (v_1, v_3) \in E'$ and $v_2 \neq v_3$, then $\mathcal{L}_{(v_1,v_2)} \cap \mathcal{L}_{(v_1,v_3)} \cap \mathcal{L}'_{v_1} = \emptyset$, and $\sum_{v \in V} w'(v, s_v, \mathcal{L}'_v) + \sum_{(v_1,v_2) \in E'} w((v_1, v_2), s_{v_1}, s_{v_2}, \mathcal{L}_{(v_1,v_2)}) + \sum_{(v_1,v_2) \in E-E'} \overline{w}((v_1, v_2), s_{v_1}, s_{v_2}, \emptyset)$ is minimum.



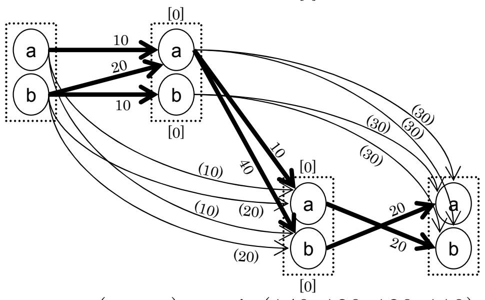
$w_1(a,a) = 140$
$w_1(a,b) = 140$
$w_1(b,a) = 160$
$w_1(b,b) = 160$



$w_2(a,a) = 120$
$w_2(a,b) = 100$
$w_2(b,a) = 110$
$w_2(b,b) = 130$



$w_3(a,a) = 120$
$w_3(a,b) = 120$
$w_3(b,a) = 130$
$w_3(b,b) = 130$

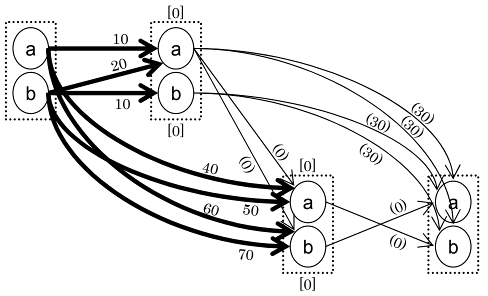


$w_4(a,a) = 110$
$w_4(a,b) = 80$
$w_4(b,a) = 130$
$w_4(b,b) = 100$

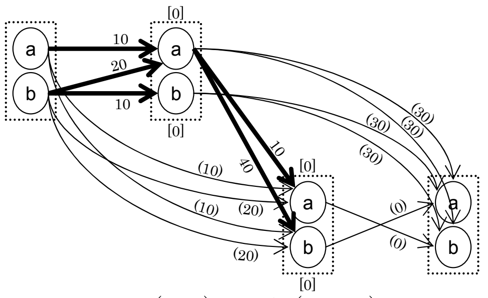
$w(e,a,a) = \min(140, 120, 120, 110) = 110$
$w(e,a,b) = \min(140, 100, 120, 80) = 80$
$w(e,b,a) = \min(160, 110, 130, 130) = 110$
$w(e,b,b) = \min(160, 130, 130, 100) = 100$

Figure 20: The trees spanning the four vertices.

For $e \in E$ and $(c_1, c_2, \mathcal{L}) \in L_e$, we call $\overline{w}(e, c_1, c_2, \mathcal{L})$ the *hidden weight* of the link $(c_1, c_2, \mathcal{L})$ of e. Given a problem instance of the original problem, if we replace each link (c_1, c_2, l) with $(c_1, c_2, \{l\})$, add a new link $(c_1, c_2, \emptyset)$ to $L_{(v_1,v_2)}$ for each $(v_1, v_2) \in E'$ and $(c_1, c_2) \in C_{v_1} \times C_{v_2}$, and set $w'(v, c, \mathcal{L}) = 0$ for all $v \in V$, $c \in C_v$ and $\mathcal{L} \subseteq \mathbb{L}$, $w(e, c_1, c_2, \{l\}) = w(e, c_1, c_2, l)$ and $\overline{w}(e, c_1, c_2, \{l\}) = \infty$ for all $e \in E$ and $(c_1, c_2, \{l\}) \in L_e$, and $w(e, c_1, c_2, \emptyset) = \infty$ and $\overline{w}(e, c_1, c_2, \emptyset) = 0$ for all $e \in E$ and $(c_1, c_2) \in C_{v_1} \times C_{v_2}$,



$$\overline{w}_1(a,a) = 80$$
$$\overline{w}_1(a,a) = 80$$
$$\overline{w}_1(a,a) = 90$$
$$\overline{w}_1(a,a) = 90$$



$$\overline{w}_2(a,a) = 60$$
$$\overline{w}_2(a,a) = 60$$
$$\overline{w}_2(a,a) = 70$$
$$\overline{w}_2(a,a) = 70$$

$$\overline{w}(a,a) = \min(80,60) = 60$$
$$\overline{w}(a,b) = \min(80,60) = 60$$
$$\overline{w}(b,a) = \min(90,70) = 70$$
$$\overline{w}(b,b) = \min(90,70) = 70$$

Figure 21: The trees spanning the three vertices.

then the problem is almost same as the original problem except that there exists at least one link between each pair of candidates (the link of empty link-label set). In the generalized problem, a minimum spanning tree always exists. If no minimum spanning tree exists in the original problem instance, then the weight of minimum spanning trees becomes infinity.

Every time when a graph is reduced, we change the weights of vertices and edges as follows:

1. In case the graph has an vertex v_2 with exactly one incoming edge (v_1, v_2) and no outgoing edges: Remove the vertex v_2 and the edge (v_1, v_2); for each $c_1 \in C_{v_1}$ and $\mathcal{L}_1 \subseteq \mathbb{L}$, copy $w''(v_1, c_1, \mathcal{L}_1) = w'(v_1, c_1, \mathcal{L}_1)$; and, for each $c_1 \in C_{v_1}$ and $\mathcal{L}_1 \subseteq \mathbb{L}$, set
$$w'(v_1, c_1, \mathcal{L}_1) = \min_{c_2 \in C_{v_2}, \mathcal{L}'_1 \subseteq \mathcal{L}_1, \mathcal{L}_2 \subseteq \mathbb{L}} (w'(v_2, c_2, \mathcal{L}_2) + w''(v_1, c_1, \mathcal{L}'_1) + w((v_1, v_2), c_1, c_2, \mathcal{L}_1 - \mathcal{L}'_1)).$$

2. In case the graph has an vertex v_2 with exactly one incoming edge (v_1, v_2) and one outgoing edge (v_2, v_3): Remove the vertex v_2 and the edges $(v_1, v_2), (v_2, v_3)$; add a new edge e from v_1 to v_3; and, for each $c_1 \in C_{v_1}$, $c_3 \in C_{v_3}$ and $\mathcal{L}_1 \subseteq \mathbb{L}$, set
$$w(e, c_1, c_3, \mathcal{L}_1) = \min_{c_2 \in C_{v_2}, \mathcal{L}_2 \subseteq \mathbb{L}, \mathcal{L}'_2 \subseteq \mathcal{L}_2} (w'(v_2, c_2, \mathcal{L}'_2) + w((v_1, v_2), c_1, c_2, \mathcal{L}_1) + w((v_2, v_3), c_2, c_3, \mathcal{L}_2 - \mathcal{L}'_2))$$
and
$$\overline{w}(e, c_1, c_3, \mathcal{L}_1) = \min_{c_2 \in C_{v_2}, \mathcal{L}_2 \subseteq \mathbb{L}, \mathcal{L}'_2 \subseteq \mathcal{L}_2} (w'(v_2, c_2, \mathcal{L}'_2) + w((v_1, v_2), c_1, c_2, \mathcal{L}_1) + \overline{w}((v_2, v_3), c_2, c_3, \mathcal{L}_2 - \mathcal{L}'_2)).$$

3. In case the graph has two vertices v_2, v_3 such that v_2 has exactly one incoming edge (v_1, v_2) and two outgoing edges $(v_2, v_3), (v_2, v_4)$, and v_3 has exactly one incoming edge (v_1, v_3) and one outgoing edge (v_3, v_4): Remove the vertices v_2, v_3 and the edges $(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_2, v_4), (v_3, v_4)$; add a new edge e from v_1 to v_4; and, for each $c_1 \in C_{v_1}$, $c_4 \in C_{v_4}$ and $\mathcal{L}_1 \subseteq \mathbb{L}$, set
$$w(e, c_1, c_4, \mathcal{L}_1) = \min(w_1(c_1, c_4, \mathcal{L}_1), w_2(c_1, c_4, \mathcal{L}_1), w_3(c_1, c_4, \mathcal{L}_1), w_4(c_1, c_4, \mathcal{L}_1))$$
and
$$\overline{w}(e, c_1, c_2, \mathcal{L}_1) = \min(\overline{w_1}(c_1, c_4, \mathcal{L}_1), \overline{w_2}(c_1, c_4, \mathcal{L}_1))$$
where:
$$\begin{aligned} w_1(c_1, c_4, \mathcal{L}_1) = &\min_{c_2 \in C_{v_2}, c_3 \in C_{v_3}, \mathcal{L}'_1 \subseteq \mathcal{L}_1, \mathcal{L}_2 \subseteq \mathbb{L}, \mathcal{L}'_2 \subseteq \mathcal{L}_2, \mathcal{L}''_2 \subseteq \mathcal{L}'_2, \mathcal{L}_3 \subseteq \mathbb{L}, \mathcal{L}'_3 \subseteq \mathcal{L}_3} (\\ &w'(v_2, c_2, \mathcal{L}''_2) + w'(v_3, c_3, \mathcal{L}'_3) + \\ &w((v_1, v_2), c_1, c_2, \mathcal{L}'_1) + \\ &w((v_1, v_3), c_1, c_3, \mathcal{L}_1 - \mathcal{L}'_1) + \\ &\overline{w}((v_2, v_3), c_2, c_3, \mathcal{L}'_2 - \mathcal{L}''_2) + \\ &w((v_2, v_4), c_2, c_4, \mathcal{L}_2 - \mathcal{L}'_2) + \\ &\overline{w}((v_3, v_4), c_3, c_4, \mathcal{L}_3 - \mathcal{L}'_3)), \end{aligned}$$
$$\begin{aligned} w_2(c_1, c_4, \mathcal{L}_1) = &\min_{c_2 \in C_{v_2}, c_3 \in C_{v_3}, \mathcal{L}'_1 \subseteq \mathcal{L}_1, \mathcal{L}_2 \subseteq \mathbb{L}, \mathcal{L}'_2 \subseteq \mathcal{L}_2, \mathcal{L}''_2 \subseteq \mathcal{L}'_2, \mathcal{L}_3 \subseteq \mathbb{L}, \mathcal{L}'_3 \subseteq \mathcal{L}_3} (\\ &w'(v_2, c_2, \mathcal{L}''_2) + w'(v_3, c_3, \mathcal{L}'_3) + \\ &w((v_1, v_2), c_1, c_2, \mathcal{L}'_1) + \\ &w((v_1, v_3), c_1, c_3, \mathcal{L}_1 - \mathcal{L}'_1) + \\ &\overline{w}((v_2, v_3), c_2, c_3, \mathcal{L}'_2 - \mathcal{L}''_2) + \\ &\overline{w}((v_2, v_4), c_2, c_4, \mathcal{L}_2 - \mathcal{L}'_2) + \\ &w((v_3, v_4), c_3, c_4, \mathcal{L}_3 - \mathcal{L}'_3)), \end{aligned}$$
$$\begin{aligned} w_3(c_1, c_4, \mathcal{L}_1) = &\min_{c_2 \in C_{v_2}, c_3 \in C_{v_3}, \mathcal{L}'_1 \subseteq \mathcal{L}_1, \mathcal{L}_2 \subseteq \mathbb{L}, \mathcal{L}'_2 \subseteq \mathcal{L}_2, \mathcal{L}''_2 \subseteq \mathcal{L}'_2, \mathcal{L}_3 \subseteq \mathbb{L}, \mathcal{L}'_3 \subseteq \mathcal{L}_3} (\\ &w'(v_2, c_2, \mathcal{L}''_2) + w'(v_3, c_3, \mathcal{L}'_3) + \\ &w((v_1, v_2), c_1, c_2, \mathcal{L}'_1) + \\ &\overline{w}((v_1, v_3), c_1, c_3, \mathcal{L}_1 - \mathcal{L}'_1) + \\ &w((v_2, v_3), c_2, c_3, \mathcal{L}'_2 - \mathcal{L}''_2) + \\ &w((v_2, v_4), c_2, c_4, \mathcal{L}_2 - \mathcal{L}'_2) + \\ &\overline{w}((v_3, v_4), c_3, c_4, \mathcal{L}_3 - \mathcal{L}'_3)), \end{aligned}$$
$$\begin{aligned} w_4(c_1, c_4, \mathcal{L}_1) = &\min_{c_2 \in C_{v_2}, c_3 \in C_{v_3}, \mathcal{L}'_1 \subseteq \mathcal{L}_1, \mathcal{L}_2 \subseteq \mathbb{L}, \mathcal{L}'_2 \subseteq \mathcal{L}_2, \mathcal{L}''_2 \subseteq \mathcal{L}'_2, \mathcal{L}_3 \subseteq \mathbb{L}, \mathcal{L}'_3 \subseteq \mathcal{L}_3} (\\ &w'(v_2, c_2, \mathcal{L}''_2) + w'(v_3, c_3, \mathcal{L}'_3) + \\ &w((v_1, v_2), c_1, c_2, \mathcal{L}'_1) + \\ &\overline{w}((v_1, v_3), c_1, c_3, \mathcal{L}_1 - \mathcal{L}'_1) + \\ &w((v_2, v_3), c_2, c_3, \mathcal{L}'_2 - \mathcal{L}''_2) + \\ &\overline{w}((v_2, v_4), c_2, c_4, \mathcal{L}_2 - \mathcal{L}'_2) + \\ &w((v_3, v_4), c_3, c_4, \mathcal{L}_3 - \mathcal{L}'_3)) \end{aligned}$$

and

$$\overline{w}_1(c_1, c_4, \mathcal{L}_1) = \min_{c_2 \in C_{v_2}, c_3 \in C_{v_3}, \mathcal{L}'_1 \subseteq \mathcal{L}_1, \mathcal{L}_2 \subseteq \mathbb{L}, \mathcal{L}'_2 \subseteq \mathcal{L}_2, \mathcal{L}''_2 \subseteq \mathcal{L}'_2, \mathcal{L}_3 \subseteq \mathbb{L}, \mathcal{L}'_3 \subseteq \mathcal{L}_3} (w'(v_2, c_2, \mathcal{L}''_2) + w'(v_3, c_3, \mathcal{L}'_3) + w((v_1, v_2), c_1, c_2, \mathcal{L}'_1) + w((v_1, v_3), c_1, c_3, \mathcal{L}_1 - \mathcal{L}'_1) + \overline{w}((v_2, v_3), c_2, c_3, \mathcal{L}'_2 - \mathcal{L}''_2) + \overline{w}((v_2, v_4), c_2, c_4, \mathcal{L}_2 - \mathcal{L}'_2) + \overline{w}((v_3, v_4), c_3, c_4, \mathcal{L}_3 - \mathcal{L}'_3)),$$

$$\overline{w}_2(c_1, c_4, \mathcal{L}_1) = \min_{c_2 \in C_{v_2}, c_3 \in C_{v_3}, \mathcal{L}'_1 \subseteq \mathcal{L}_1, \mathcal{L}_2 \subseteq \mathbb{L}, \mathcal{L}'_2 \subseteq \mathcal{L}_2, \mathcal{L}''_2 \subseteq \mathcal{L}'_2, \mathcal{L}_3 \subseteq \mathbb{L}, \mathcal{L}'_3 \subseteq \mathcal{L}_3} (w'(v_2, c_2, \mathcal{L}''_2) + w'(v_3, c_3, \mathcal{L}'_3) + w((v_1, v_2), c_1, c_2, \mathcal{L}'_1) + \overline{w}((v_1, v_3), c_1, c_3, \mathcal{L}_1 - \mathcal{L}'_1) + w((v_2, v_3), c_2, c_3, \mathcal{L}'_2 - \mathcal{L}''_2) + \overline{w}((v_2, v_4), c_2, c_4, \mathcal{L}_2 - \mathcal{L}'_2) + \overline{w}((v_3, v_4), c_3, c_4, \mathcal{L}_3 - \mathcal{L}'_3)),$$

4. In case the graph has multiple edges e, e' from v_1 to v_2 (caused by the applications of the reduction rule 2 or 3): Replace them with a new edge e''; and, for each $c_1 \in C_{v_1}$, $c_2 \in C_{v_2}$ and $\mathcal{L}_1 \subseteq \mathbb{L}$, set

$$w(e'', c_1, c_2, \mathcal{L}_1) = \min(\min_{\mathcal{L}'_1 \subseteq \mathcal{L}_1} (w(e, c_1, c_2, \mathcal{L}'_1) + \overline{w}(e', c_1, c_2, \mathcal{L}_1 - \mathcal{L}'_1)), \min_{\mathcal{L}'_1 \subseteq \mathcal{L}_1} (\overline{w}(e, c_1, c_2, \mathcal{L}'_1) + w(e', c_1, c_2, \mathcal{L}_1 - \mathcal{L}'_1)))$$

and

$$\overline{w}(e'', c_1, c_2, \mathcal{L}_1) = \min_{\mathcal{L}'_1 \subseteq \mathcal{L}_1} (\overline{w}(e, c_1, c_2, \mathcal{L}'_1) + \overline{w}(e', c_1, c_2, \mathcal{L}_1 - \mathcal{L}'_1)).$$

In all the cases, some weights of links and candidates with a link-label set are selected, and the other weights are abandoned. If we remember which weights of links and candidates with a link-label set are surviving, we can obtain a minimum spanning tree of the original graph with a proper selection of candidates.

4.4. Time Complexity of the Solutions

We will summarize the time complexity of the solutions introduced in this section.

For the solution to MSTP for DAGs, the time complexity is $O(n)$, where n is the number of vertices of a graph. The number of reduction step of a graph is at most $2 \cdot n$. Every time when a graph is reduced, the weight of the reduced graph can be calculated in a constant time.

For the solution to MSTP with candidate selection, the time complexity is $O(n \cdot m^4)$, where n is the number of vertices of a graph, and m is the maximum number of candidates of a vertices. Since the weight of a new edge needs to be calculated for each combination of candidates, the time complexity is m^4 times greater. However, m can be considered as a constant number for the application to mathematical OCR. The number of character recognition candidates is at most 10 in most cases. Therefore, the time complexity can be seen as $O(n)$.

For the solution to MSTP with candidate and link-label selections, the time complexity is $O(n \cdot m^4 \cdot (2^l)^7)$, where n is the number of vertices of a graph, m is the maximum number of candidates of a vertices, and l is the number of link-labels. Fortunately, $l = 7$ for the application to mathematical OCR, namely, “horizontal”, “superscript”, “subscript”, “upper”, “under”, “left superscript” and “left subscript.” Moreover, $(2^l)^7$ can be reduced to $(2^l)^5$ because some values can be calculated independently. $2^{35} = 34,359,738,368$ is not too big for personal computers of today. In addition, this number is the worst case. We can get the answer with a much less number of calculations in real cases. Therefore, the time complexity can be seen as $O(n)$.

5. Discussion

In the previous section, we have seen that, for DAGs in the class RD, MSTP with candidate and link-label selections can be solved in linear-time in the number of vertices of a graph. In this section, we discuss the usefulness of the solution for the application to mathematical OCR.

5.1. Coverage of Mathematical Formulae with the Class RD

To claim the usefulness of the solution, we need to investigate the coverage of mathematical formulae with the class RD. 21,742 mathematical formulae are collected from 31 pure mathematical articles, and virtual link networks are generated by InftyReader [11], a mathematical OCR system. Since 222 virtual link networks are not single-source DAGs, we use 21,520 virtual link networks for this investigation.

On Table 1, the coverage of mathematical formulae with the class RD and the graph classes of a bounded treewidth are shown. We find that most virtual link networks have treewidth at most 3.

Table 1: Coverage of Mathematical Formulae with Graph Classes.

Graph Class	Number	Coverage Rate
Class RD	18,980	88.2 %
Treewidth= 1	8,755	40.7 %
Treewidth≤ 2	17,598	81.8 %
Treewidth≤ 3	20,795	96.6 %
Total	21,520	100 %

In Figrue 22, the Venn diagrams which explain the relation between the class RD and treewidth is shown. Though the class RD covers all single-source DAGs of treewidth 1, 345 virtual link networks of treewidth 2 and 1,470 virtual link networks of treewidth 3 are not covered. The class RD does not cover some simple graphs. Examples of simple graphs not in the class RD are shown in Figrue 23.

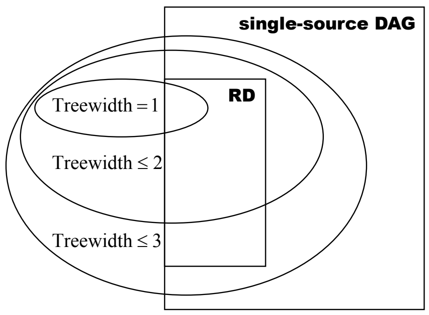


Figure 22: Relation between the Class RD and Treewidth.

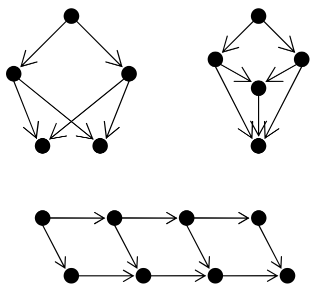


Figure 23: Graphs not in the class RD.

5.2. Solutions for Graphs not in the Class RD

Though the coverage rate of the class RD 88.2% is satisfactory, we need to have solutions for graphs not in the class RD. We think the following two solutions are effective:

- Elimination of edges until a graph becomes a member of the class RD, and
- Enumeration of all spanning trees of a reduced graph.

Obviously, elimination of edges makes any graph be a member of the class RD. If we heuristically eliminate useless edges and transform it to a graph in the class RD, then we could obtain a spanning tree whose weight is close to the minimum.

By using the reduction rules of the class RD, though some graphs are not reduced to a single-vertex graph, most of them can be reduced to a simple graph. The second solution takes advantage of this feature. If the number of spanning trees of a reduced graph is less than a fixed constant, then we enumerate all spanning trees and calculate weights of them. The minimum weight of them is exactly the weight of the minimum spanning trees of the original graph.

By combining the above solutions for graphs not in the class RD, we have a solution to MSTP with candidate and link-label selections for the general case.

6. Conclusion

We introduced a variation of the minimum spanning tree problem for the application to mathematical OCR. Unfortunately, the problem was shown to be NP-hard. However, we introduced a class of DAGs that is recursively defined with some graph-rewriting rules so that it contains only graphs with a bounded treewidth. For graphs in the class RD, it was shown that the problem can be solved in linear-time in the number of vertices of a graph.

By an investigation on 21,520 mathematical formulae from 31 pure mathematical articles, we found that the class RD covers 88.2% of mathematical formulae. For graphs not in the class RD, we suggested solutions for them.

References

[1] Stefan Arnborg and Andrzej Proskurowski, *Linear time algorithms for NP-hard problems restricted to partial k-trees*, Discrete Appl. Math. **23** (1989), no. 1, 11–24.

[2] Hans L. Bodlaender, *A linear time algorithm for finding tree-decompositions of small treewidth*, STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing (New York, NY, USA), ACM, 1993, pp. 226–234.

[3] Hans L. Bodlaender, *A partial k-arboretum of graphs with bounded treewidth*, Theor. Comput. Sci. **209** (1998), no. 1-2, 1–45.

[4] Kam-Fai Chan and Dit-Yan Yeung, *Mathematical expression recognition: a survey*, Int. J. Document Analysis and Recoginition **3** (2000), no. 1, 3–15.

[5] Ruay-Shiung Chang and Shing-Jiuan Leu, *The minimum labeling spanning trees*, Inf. Process. Lett. **63** (1997), no. 5, 277–282.

[6] Yuko Eto and Masakazu Suzuki, *Mathematical formula recognition using virtual link network*, Proceedings of the 6th International Conference on Document Analysis and Recognition (ICDAR 2001), 2001, pp. 430–437.

[7] Joseph B. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proceedings of the American Mathematical Society **7** (1956), no. 1, 48–50.

[8] Kazuhisa Makino, Yushi Uno, and Toshihide Ibaraki, *On minimum edge ranking spanning trees*, J. Algorithms **38** (2001), no. 2, 411–437.

[9] Young-Soo Myung, Chang-Ho Lee, and Dong-Wan Tcha, *On the generalized minimum spanning tree problem*, Networks **26** (1995), no. 4, 231–241.

[10] R. C. Prim, *Shortest connection networks and some generalizations*, Bell System Technology Journal **36** (1957), 1389–1401.

[11] Masakazu Suzuki, Fumikazu Tamari, Ryoji Fukuda, Seiichi Uchida, and Toshihiro Kanahori, *Infty - an integrated OCR system for mathematical documents*, Proceedings of ACM Symposium on Document Engineering 2003, 2003, pp. 95–104.

Akio Fujiyoshi
Department of Computer and Information Sciences, Ibaraki University, Hitachi 316-8511, Japan.
E-mail: fujiyosi(at)mx.ibaraki.ac.jp

Masakazu Suzuki
Graduate School of Mathematics, Kyushu University, Fukuoka 812-8581, Japan.
E-mail: suzuki(at)math.kyushu-u.ac.jp