アルゴリズム　20－２
（１９９１．３．１５）

最小ｒａｎｇｅカット問題の効率的解法

加藤　直樹*　　岩野　和生†

＊神戸商科大学　管理科学科　　†日本ＩＢＭ　東京基礎研究所

$G=(V,E)$ を n 個の節点と m 本の枝からなる連結な無向グラフとし、各枝 e に重み $w(e)$ が与えられているものとする。本論文は G のカットのなかでそのカットに属する枝の重みの中でその最大値と最小値の差（それをｒａｎｇｅと呼ぶ）を最小にするカットを求める問題を扱い、$O(MST(m,n)+n\log n)$ の算法を提案する。ここで $MST(m,n)$ は G の最小木及び最大木を求める手間を表す。またあらかじめターゲットと呼ばれる値が与えられていてカットに属する枝の重みの中でその最大値と最小値の間にターゲットを含むカットのなかでｒａｎｇｅを最小にするカットを求める問題も扱い、同じ計算手間を持つ算法を提案する。

Efficient Algorithms for the Minimum Range Cut Problems

Naoki Katoh*, Kazuo Iwano†

* Department of Management Science, Kobe University of Commerce
Gakuen-Nishimachi 8-2-1, Nishi-ku, Kobe 651-21, Japan

†Tokyo Research Laboratory, IBM Japan
5-11 Sanbancho, Chiyoda-ku, Tokyo 102, Japan

Let $G=(V,E)$ be a connected undirected graph with n vertices and m edges, in which a real-valued *weight*, denoted by $w(e)$, is associated with every edge e. This paper studies the problem that asks to find a cut in G such that the range of all edge weights in the cut is minimum. Here the range of a cut C is defined to be the maximum difference among weights of edges in the cut, i.e., $\max_{e\in C} w(e) - \min_{e\in C} w(e)$. We call this problem the *minimum range cut problem.* This paper presents an $O(MST(m,n)+n\log n)$ algorithm for this problem, where $MST(m,n)$ denotes the time required to compute minimum and maximum spanning trees of G. We then consider the minimum range target cut problem which asks to find a minimum range cut such that the interval of edge weights therein (i.e., $[\min_{e\in C} w(e), \max_{e\in C} w(e)]$) contains a prespecified value (called *target*). We call this problem the . We show that this problem can also be solved in a manner similar to the above problem with the same time complexity.

1 Introduction

Let $G = (V, E)$ be a connected undirected graph with n vertices and m edges, in which a real-valued *weight*, denoted by $w(e)$, is associated with every edge e. This paper studies the problem that asks to find a cut in G such that the range of all edge weights in the cut is minimum. Here a cut in G is a subset of E whose removal from G disconnects G, and the *range* of a cut is defined to be the maximum difference among weights of edges in the cut, i.e., $\max_{e\in C} w(e) - \min_{e\in C} w(e)$. We call this problem the *minimum range cut problem.* Recently several researchers studied a number of minimum range problems, e.g., minimum range spanning tree problems by [1], minimum range assignment problems by [6], and so on. To the authors' knowledge, no one has ever considered the minimum range cut problem.

We shall propose in this paper an $O(MST(m,n) + n\log n)$ algorithm for the minimum range cut problem, where $MST(m,n)$ denotes the time required to compute minimum and maximum spanning trees of G. The best known bound for $MST(m,n)$ is $O(m \log \beta(m,n))$ by Gabow at al. [4], where $\beta(m,n)$ is defined to be $\min\{i \mid \log^{(i)} n \le m/n\}$, and $\log^{(i)} x$ denotes the log function iterated i times.

The first important observation is that only the edges of minimum and maximum spanning trees are sufficient to find a minimum range cut. Thus, after preprocessing the original graph by computing minimum and maximum spanning trees, the edge set E can be reduced to the set of at most $2(n-1)$ edges. This result is rather surprising because a closely related problem, the minimum range spanning tree problem, does not have such a property.

For a closed interval $[\alpha, \beta]$ with $\alpha \le \beta$, suppose that there is a cut C such that all edge weights of C lie in the interval. If there is no closed interval $[\alpha', \beta']$ properly included in $[\alpha, \beta]$ that satisfies the above property, $[\alpha, \beta]$ is called *critical.* Such a cut C is called a *critical cut.* It is clear that the minimum range cut problem can be reduced to the problem of finding a critical interval $[\alpha, \beta]$ such that $\beta - \alpha$ is minimum. Thus, a naive approach is to enumerate all critical intervals, and then identify a minimum range cut as a critical cut in the critical interval with the minimum range. Enumerating all critical intervals is done in our paper by following the systematic approach proposed by *Martello et al.* [6]. Though simply applying their algorithm to our problem requires $O(n^2)$ time, the algorithm in this paper accelerates their algorithm by making use of the special structure of our problem. A critical interval can be characterized in our problem by a maximum spanning tree of a graph G in which edge weights are appropriately modified. With this characterization, given a critical interval, a next critical interval can be efficiently computed by updating the current maximum spanning tree by using dynamic trees proposed by Sleator and Tarjan [9]. This approach leads to an $O(MST(m,n) + n\log n)$ time algorithm.

If G is planar, a minimum range cut can be obtained in $O(n \log n)$ time because $MST(m,n) = O(n)$ for planar graphs [2]. If G is Euclidean, i.e., V is the set of points in the plane and the weight of an edge is equal to the distance between the corresponding vertices, both minimum and maximum spanning trees can be obtained in $O(n\log n)$ time [7], [8]. A minimum range cut is thus obtained in $O(n \log n)$ time. Notice that, in case G is Euclidean, the output of the algorithm is not the set of edges in the cut, but a partition of vertices induced by the cut, because reporting the set of edges in the cut may require $O(n^2)$ time.

This paper then studies the problem of finding a minimum range cut with a target, which asks to find a minimum range cut such that the interval of edge weights therein (i.e., $[\min_{e\in C} w(e), \max_{e\in C} w(e)]$) contains a prespecified value (called *target*). We call this problem the *minimum range target cut problem.* The algorithm for the previous problem cannot be applied to this problem because no critical interval computed by the algorithm for the previous problem may contain a target value. However, we can show that this problem can also be solved in a manner similar to the above problem with the same time complexity.

This paper is organized as follows. Section 2 reviews previous work on minimum range problems. Section 3 presents an $O(MST(m,n) + n\log n)$ algorithm for the minimum range cut problem. Section 4 studies the minimum range target cut problem and develops an $O(MST(m,n) + n \log n)$ algorithm.

2 Previous work

We begin this section with presenting a formal description of a general class of minimum range problems given by Martello et al. [6]. Suppose that we are given a finite set E, a family $\mathcal{F}$ of *"feasible subsets"* of E and a real-valued weight $w(e)$ associated with every edge $e \in E$. The minimum range problem (also called the *balanced optimization problem* by [6]) is then described as follows:

$$\text{minimize}_{S\in\mathcal{F}} \max\{w(e) - w(e') \mid e, e' \in S\}. \tag{1}$$

In other words, this problem tries to make the difference in value between the largest and smallest weights used as small as possible. Martello et al. [6] showed that, if an efficient feasibility subroutine is available, then we can efficiently solve this problem.

A *feasibility subroutine* accepts as input a subset $E' \subseteq E$ and either produces some $S \in \mathcal{F}$ with $S \subseteq E'$ or else states that no such S exists. For the minimum range cut problem, $\mathcal{F}$ stands for the set of all cuts in G, and a feasibility subroutine simply determines whether $G = (V, E - E')$ is disconnected.

Let $v_1 < v_2 < \cdots < v_p$ be the sorted list of all the distinct values in $\{w(e) \mid e \in E\}$. Since there are $O(p^2)$ distinct intervals $[v_i, v_j]$, a naive approach that simply applies a feasibility subroutine to $\{e \mid v_i \leq w(e) \leq v_j\}$ for all possible i and j requires $O(p^2 f(|E|))$ time, where $f(|E|)$ denotes the time required for the feasibility subroutine.

Martello et al. [6] improved this time bound to $O(pf(|E|))$ for the general minimum range problem, and gave an $O(n^4)$ algorithm when $\mathcal{F}$ is the set of all assignments in bipartite graphs. Camerini at al. [1] studied the case that $\mathcal{F}$ is the set of all spanning trees in undirected and directed graphs (they call *uniform spanning tree problems*), and gave $O(mn)$ and $O(m^2)$ algorithms respectively. Camerini at al. [1] also studied variants of these problems. Galil and Schieber [5] improved $O(mn)$ algorithm in [1] for the undirected case to $O(m \log n)$ by making use of *dynamic* trees of Sleator and Tarjan [9].

3 Minimum Range Cut Problem

Since it takes $O(m)$ time to test the connectivity of a given graph, our problem can be solved in $O(m^2)$ time by simply applying the algorithm in [6] with $p = O(m)$. In this section, we improve this trivial time bound to $O(MST(m,n) + n \log n)$.

Let (a, b) denote the interval $\{x \mid a < x < b\}$. When we include a boundary of the interval, we use "[" (*resp.* "]") instead of "(" (*resp.* ")", for example, $[a, b) = \{x \mid a \leq x < b\}$. Given an interval $[a, b)$, we define $E[a, b)$ as $\{e \in E \mid a \leq w(e) < b\}$. $E(a, b]$ is similarly defined. For the simplicity we use $E(a)$ instead of $E[a, a]$.

A cut C is $[a, b]$-*critical* when the minimum (*resp.* maximum) weight of edges in C is a (*resp.* b), and there is no cut in $E(a, b]$ or $E[a, b)$. A value a is said to be *lower-critical* if there exists a $[a, b]$-critical cut C for some b with $a \leq b$. Analogously, we define an *upper-critical* value. A closed interval $[a, b]$ is *critical* if there exists a $[a, b]$-critical cut. For a cut C, let $\max(C) \equiv \max_{e\in C} w(e)$, $\min(C) \equiv \min_{e\in C} w(e)$, and *max-edge(C)* (resp. *min-edge(C)*) denotes the edge $e \in C$ with $w(e) = max(C)$ (resp. $w(e) = min(C)$). We use the following facts in our algorithm.

Fact 1. If there exists a $[a, b]$-critical cut, then there exists a spanning tree in $E - E[a, b)$ and $E - E(a, b]$, but not in $E - E[a, b]$. □

Fact 2. A minimum range cut is critical to some interval. □

Since a critical interval cannot be contained in any other critical interval, we can order critical intervals by their lower boundaries. We say that a critical interval $[a_1, b_1]$ is *lower* (*resp. higher*) than a critical interval $[a_2, b_2]$ when $a_1 < a_2$ (*resp.* $a_1 > a_2$).

Let T_{max} (*resp.* T_{min}) denote a maximum (*resp.* minimum) spanning tree of G. We use the convention throughout the paper that T_{max} (*resp.* T_{min}) represents the set of edges in T_{max} (*resp.* T_{min}). The following Lemma shows that we only have to process the edges of $T_{max} \cup T_{min}$ in order to compute all critical intervals. That is, after obtaining maximum and minimum spanning trees of G, we can reduce the edge set E to $T_{max} \cup T_{min}$.

Lemma 1. For any cut C, there exists an edge $e \in T_{max} \cap C$ (*resp.* $e \in T_{min} \cap C$) with $w(e) = \max(C)$ (*resp.* $w(e) = \min(C)$).

Proof. Let $w(e') = \max\{w(e) \mid e \in T_{max} \cap C\}$, and let *max-edge*$(C) = (u, v)$. Suppose $w(e') < max(C)$. Since *max-edge*$(C) \notin T_{max}$, there exist a path $p(u, v)$ from u to v on T_{max} and an edge $\hat{e} \in p(u, v) \cap C$. Since T_{max} is a maximum spanning tree, $w(\hat{e}) \geq \max(C)$ follows. This contradicts $w(e') < \max(C)$. Thus, there exists an edge $e \in T_{max} \cap C$ with $w(e) = \max(C)$. Similarly, we can show that there exists an edge $e \in T_{min} \cap C$ with $w(e) = \min(C)$. □

¿From the above lemma, our algorithm first compute a minimum and maximum spanning tree T_{min} and T_{max}, and then reduces the edge set E to $T_{min} \cup T_{max}$. For the simplicity, we assume throughout the succeeding discussion that E is already reduced to $T_{min} \cup T_{max}$.

Our algorithm enumerates critical intervals $[a_1, b_1]$, $[a_2, b_2], \ldots$ in the increasing order of their lower boundaries. The general scheme of our algorithm follows the algorithm by Martello et al. [6]. However, we elaborate on that algorithm in order to speed up the computation of the next higher critical interval $[a_{i+1}, b_{i+1}]$ when the current critical interval $[a_i, b_i]$ is given. The following Lemma shows how to compute b_1.

Lemma 2. The minimum edge-weight β of a maximum spanning tree is the lowest upper-critical value.

Proof. For any $x \leq \beta$, there exists a spanning tree in $E[x, v_p]$. □

Since $E[v_1, b_1]$ has a cut but $E(a_1, b_1]$ does not have a cut from the definition of a_1, the lower boundary a_1 is characterized as the minimum value x such that $G' = (V, E[v_1, x] \cup E(b_1, v_p])$ is connected. It is computed by starting with graph $G' = (V, E(b_1, v_p])$, and then adding the edges of $E(v_1)$, $E(v_2)$, $\ldots$ to G' in this order until the augmented graph first becomes connected. During this process, we maintain connected components, which are represented as a spanning forest, of the corresponding graph. When a_1 is obtained, we have a spanning tree in $G = (V, E[v_1, a_1] \cup E(b_1, v_p])$.

After obtaining the lowest critical interval $[a_1, b_1]$, the next higher critical interval $[a_2, b_2]$ is computed. This is done in general as follows. The algorithm maintains the *current* critical interval $[\alpha, \beta]$, the graph $G_{\alpha,\beta} \equiv (V, E[v_1, \alpha] \cup E(\beta, v_p])$, and a spanning tree T in $G_{\alpha,\beta}$. The computation of the next higher critical interval $[\alpha', \beta']$ consists of the following two phases; that is, (1) the first phase computes the next higher upper-critical value β', and (2) the second phase computes the next higher lower-critical value α' when given an associated upper-critical value β'.

We use the following properties to compute α' or β' in each phase. Since β' is an upper-critical value next higher than β, there is a cut in $E(\alpha, \beta']$ but there does not exist a cut in $E(\alpha, \gamma]$ for $\gamma < \beta'$. Therefore, β' is the minimum value y such that $E[v_1, \alpha] \cup E(y, v_p]$ is disconnected. Since α' is the lower-critical value next higher than α, there exists a cut in $E[\alpha', \beta']$ but not in $E[\gamma, \beta']$ for $\gamma > \alpha'$. Thus, α' is the minimum value x such that $E[v_1, x] \cup E(\beta', v_p]$ is connected.

Let M be a constant such that $M > v_p$.

In the first phase, we build a spanning tree T' of maximum weight in $G_{\alpha,\beta}$ in which weights of edges in $E[v_1, \alpha]$ are modified as $w'(e) = M$ and those in $E(\beta, v_p]$ are not changed. (We call a spanning tree T' defined above *the modified maximum spanning tree in* $G_{\alpha,\beta}$.) Then β' is the minimum of modified edge weights in T', i.e., the minimum of original edge weights in $T' \cap E(\beta, v_p]$. This is justified by the following Lemma.

Lemma 3. Let $[\alpha, \beta]$ be the current critical interval and T' be a modified maximum spanning tree in $G_{\alpha,\beta}$. If $[\alpha, \beta]$ is not the highest critical interval, then the minimum of modified edge-weights in T' is the upper-critical value next higher than β.

Proof. Let y be the minimum of modified edge weights in T'. Let β' be the upper-critical value next higher than β with the associated critical interval $[\alpha', \beta']$. Since β' is upper critical, there exists an edge $e = (u, v)$ of weight β'. Suppose $\beta' < y$. Since $E[v_1, \alpha] \cup E(\beta, v_p]$ is connected and β' is an upper-critical value, β' is the minimum value such that $\beta' > \beta$ and $E[v_1, \alpha] \cup E(\beta', v_p]$ is disconnected. ¿From the definition of T', there exists a path from u to v on T' using edges of modified weights larger than β'. Therefore, u and v are connected by a path in $E[v_1, \alpha] \cup E[y, v_p]$. This contradicts to that there exists a critical cut in $(\alpha, \beta']$. □

The description of the entire algorithm is given in Figure 1. The procedure *Refine*(x, y, T) in Figure 2 executes the second phase that finds a next higher lower-critical value when a modified maximum spanning tree T in $G_{x,y}$ is given. In this phase, we first obtain a spanning forest F' of maximum modified weight in $E[v_1, \alpha] \cup E(\beta', v_p]$, (which is simply obtained from T' by deleting edge(s) of weight β'). Let $v_i = \alpha$. We then add edges of $E(v_{i+1})$, $E(v_{i+2})$, $\ldots$ with modified weight M until F' with augmented edges contains a spanning tree. Note that since augmented edges are of the maximum weight M, a spanning forest in F' with augmented edges is of maximum modified weight, and moreover, a spanning tree finally obtained is also of maximum modified weight in $E[v_1, \alpha'] \cup E(\beta', v_p]$. The role of modified weights is to ensure that edges to be added in the algorithm have a higher priority than those in $E(\beta', v_p]$ to be included in a spanning forest maintained by the algorithm.

```
Procedure Minimum-Range-Cut
begin
(1)     Compute minimum and maximum spanning trees $T_{min}$ and $T_{max}$, and
        let $E = T_{min} \cup T_{max}$.
        $\alpha = 0$; $z = v_p - v_1$;
(2)     Let $v_1, v_2, \cdots, v_p$ be the increasing sequence of distinct weights of edges
        in $E$.
        $\alpha = 0$; $z = v_p - v_1$;
(3)     $T \leftarrow T_{max}$;
(4)     do until (all edges in $T$ have weights $M$)
(4.1)   $\beta = w(e)$ such that $e \in T$ and $w'(e)$ is minimum among modified edge
        weights in $T$;
(4.2)   $(\alpha, \beta, T) \leftarrow Refine(\alpha, \beta, T)$;
(4.3)   if $z > \beta - \alpha$ then $\{z = \beta - \alpha;\ \alpha^* = \alpha;\ \beta^* = \beta;\}$
        end
(5)     Compute the connected components of $E[\alpha^*, \beta^*]$ and report a set of
        edges which connect a connected component and the others.
end
```

Figure 1. Algorithm *Minimum-Range-Cut*.

Theorem 1. We can compute a minimum range cut in $O(MST(m, n) + n \log n)$.
Proof. It is clear from Lemmas 1, 2 and 3 and from the explanation given above that the algorithm correctly computes an minimum range cut. We now analyze the time complexity. Finding T_{min} and T_{max} requires $O(MST(m, n))$ time. Sorting weights of at most $2(n-1)$ edges in T_{min} and T_{max} requires $O(n \log n)$ time. Finding the next higher upper-critical value β' can done in $O(n \log n)$ time in total by using a heap [10]. The other time-consuming part is to update a modified maximum spanning tree to find the next higher lower-critical value. This can be implemented by using dynamic trees [9]. We associate each modified edge-weight with an edge of dynamic trees. Initially, we build a dynamic tree associated with a maximum spanning tree. When we add an edge $e = (u, v)$ to T', we first check if the addition of this edge creates a cycle or not. If so, we delete a minimum weight edge on the cycle by $evert(u)$ and $cut(findmin(v))$ operations. Then we link two trees by $link(evert(u), v)$ operation (see [9] for the details of operations $link$, $evert$, and $findmin$). Since performing a sequence of n operations to the dynamic trees takes $O(n \log n)$ amortized time, it takes $O(n \log n)$ in total. □

In particular, the time complexity shown in the above theorem becomes $O(n \log n)$ for planar or Euclidean graphs, since $MST(m, n) = O(n)$ for planar graphs [2] and $MST(m, n) = O(n \log n)$ for Euclidean graphs [7], [8].

```
procedure $Refine(x, y, T)$
begin
(1)     $T' \leftarrow T - E(y)$;
(2)     Suppose $x = v_i$.
(3)     do $k = i + 1$ to $p$ until ($T'$ becomes a spanning tree)
(3.1)       add $E(v_k)$ to $T'$ and maintain $T'$ as a modified maximum spanning tree;
        end
(4)     $\alpha = v_k$;
(5)     $return(\alpha, y, T')$;
end
```

Figure 2. Procedure *Refine*.

4 Minimum Range Target Cut Problem

In this section, we consider the following variant of the minimum range cut problem. We shall show that this problem can be solved in the same time complexity as the algorithm in Section 3.

The minimum range target cut problem: Given a connected undirected graph $G = (V, E)$ with a real valued edge-weight function w and a target value γ, we find a cut with the minimum range containing γ. □

We may consider that we can solve this problem by picking up an appropriate critical interval from intervals obtained by the algorithm in the previous section. However, this approach may not work. This is because that there may be no feasible solution among critical intervals generated by the algorithm in the previous section, since γ may not be contained in any critical interval even if γ satisfies $v_1 \le \gamma \le v_p$. To overcome this difficulty, we define new concepts *upper-critical interval (or cut) with respect to* e for each $e \in T_{min}$ in the next paragraph.

First notice that as in the minimum range cut problem, the edge set E can be reduced to $T_{min} \cup T_{max}$ by Lemma 1. To avoid an complicated argument, we assume that no two edges in E have the same weight (the other case can be similarly treated). ¿From Lemma 1, we can assume without loss of generality that *max-edge*$(C) \in T_{max}$ and *min-edge*$(C) \in T_{min}$. The interval $[w(e), \beta]$ is called *upper-critical with respect to* e if there exists a cut C such that *min-edge*$(C) = e$ (i.e., $\min(C) = w(e)$) and $\max(C) = \beta$, but there does not exist a cut C' such that *min-edge*$(C') = e$ and $max(C') < \beta$. Such C is called a *upper-critical cut with respect to* e and is denoted by $C(e)$. Let *upper*(e) denote *max-edge*(C). Since all edge weights are assumed to be distinct, such C is uniquely determined. For each $f \in T_{max}$, *lower-critical interval (or cut) with respect to* f is similarly defined.

The algorithm first computes upper-critical intervals with respect to e and *upper*(e) for each $e \in T_{min}$. We will then show that, if $e \ne e'$ for $e, e' \in T_{min}$, *upper*$(e) \ne$ *upper*(e') holds. This fact is a key to finding a minimum range target cut.

Let $\mathcal{C}$ denote the set of cuts $C(e)$ for all $e \in T_{min}$. We now define the following three subsets of $\mathcal{C}$:

$$\mathcal{C}_1 = \{C(e) \in \mathcal{C} \mid \min(C(e)) \le \gamma \le \max(C(e))\}, \tag{2}$$
$$\mathcal{C}_2 = \{C(e) \in \mathcal{C} \mid \min(C(e)) > \gamma\}, \tag{3}$$
$$\mathcal{C}_3 = \{C(e) \in \mathcal{C} \mid \max(C(e)) < \gamma\}. \tag{4}$$

Let also

$$r_1 = \min\{\max(C(e)) - \min(C(e)) \mid C(e) \in \mathcal{C}_1\}, \tag{5}$$
$$r_2 = \min\{\max(C(e)) \mid C(e) \in \mathcal{C}_2\}, \tag{6}$$
$$r_3 = \max\{\min(C(e)) \mid C(e) \in \mathcal{C}_3\}. \tag{7}$$

Let e^* (*resp.* e^{**}, e^{***}) be an edge which realizes the above value r_1 (*resp.* r_2, r_3). Then a cut $C(e^*)$ is a candidate for a minimum range target cut, but cannot be concluded to be a minimum range target cut. The reason is that, for any $C(e') \in \mathcal{C}_2$ and $C(e'') \in \mathcal{C}_3$, a cut $C(e') \cup C(e'')$ clearly contains a target γ, and such a cut may have a shorter range. Among those cuts, the best candidate for a minimum range target cut is $C(e^{**}) \cup C(e^{***})$. Between these two cuts $C(e^*)$ and $C(e^{**}) \cup C(e^{***})$, the one with smaller range is output as the desired minimum range target cut (the rigorous proof will be given later in Lemma 7).

We first give the outline of the algorithm for finding all upper-critical intervals with respect to all $e \in T_{min}$. The algorithm is similar to Algorithm *Minimum-Range-Cut* given in the previous section. Let $e_1, e_2, \ldots, e_{n-1}$ be the list of edges in T_{min} with $w(e_1) < w(e_2) < \cdots < w(e_{n-1})$ (recall that we assume all edge weights are distinct). Define T_i for i with $1 \le i \le n-1$ to be a spanning tree that has the minimum weight among all spanning trees that contain $\{e_1, e_2, \ldots, e_i\}$. We use the convention that T_0 stands for T_{max}. Starting with T_0, the algorithm computes $T_1, T_2, \ldots, T_{n-1}$ in this order by performing an edge-exchange one at a time. When T_i is obtained from T_{i-1} by performing an edge-exchange (e_i, f_i) (i.e., $T_i = (T_{i-1} - f_i) \cup e_i$), *upper*$(e_i)$ is output as f_i. Since outputting the edge set of all T_i requires $O(n^2)$ time, we simply report (e_i, f_i) for each i. The algorithm is described as follows.

Step 1: Let $T = T_{min}$ and $i = 1$.

Step 2: For $e_i = (u_i, v_i)$, let f_i be the edge such that $w(f_i)$ is minimum among edges on the unique path between u_i

and v_i in T. Let $T = (T - f_i) \cup e_i$ and $w(e_i) = M$. Output f_i as *upper*(e_i). Let $i = i + 1$.
Step 3: If $i \leq n - 1$ return to Step 2. Else stop.

Since we assume that all edge weights are distinct, for each $f_i \in T_{max}$, there exists the unique e_i such that *upper*$(e_i) = f_i$. Here M used in Step 2 is a big constant defined in the previous section. A tree T newly obtained in Step 2 is equal to T_i as will be proved in Lemma 4. We shall show in Lemma 5 that $f_i =$ *upper*(e_i) holds for all i.

Lemma 4. The algorithm explained above correctly computes T_i for all i with $1 \leq i \leq n - 1$.

Proof. Note that f_i computed by the algorithm belongs to T_{max} since $w(f_i)$ is of minimum weight among all edges on the path of T_{i-1} between u_i and v_i and all edges $e_1, e_2, \ldots, e_{i-1}$ have the weight M which is larger than $w(f_i)$ from the definition of M. Then the proof is done in a straightforward manner by induction on i. Thus the details are omitted. □

Let $f_i = (x_i, y_i)$ and let $T(x_i)$ and $T(y_i)$ be the two subtrees obtained by deleting f_i from T_{i-1} such that $x_i \in T(x_i)$ and $y_i \in T(y_i)$. Let $V(x_i)$ (resp. $V(y_i)$) be the set of vertices of $T(x_i)$ (resp. $T(y_i)$). Assume without loss of generality that $u_i \in T(x_i)$ and $v_i \in T(y_i)$.

Lemma 5. For each e_i with $1 \leq i \leq n - 1$, let the cut C be the set of edges connecting $V(x_i)$ and $V(y_i)$. Then (1) C is the upper-critical cut with respect to e_i, and (2) C is the lower-critical cut with respect to f_i.

Proof. We first prove (1). Note that $e_i, f_i \in C$ holds. Suppose that $max(C(e_i)) < w(f_i)$ holds for the upper-critical cut $C(e_i)$ with respect to e_i. Thus $C(e_i)$ does not contain f_i. Since $C(e_i)$ must contain at least one edge on the path of T_{i-1} between u_i and v_i and all edges on the path have larger weights than $w(f_i)$ or smaller weights than $w(e_i)$, this is a contradiction. Therefore, $max(C(e_i)) = w(f_i)$ holds. If $max(C) > w(f_i)$, the edge *max-edge(C)* $\in T_{max}$ connects $V(x_i)$ and $V(y_i)$. Exchanging f_i and *max-edge(C)* creates a spanning tree that has a weight larger than T_{i-1} and contains $\{e_1, e_2, \ldots, e_{i-1}\}$, which contradicts Lemma 4. This proves (1). We can also prove (2) in a similar way as above. □

Note that this lemma shows that Step 2 in the algorithm correctly computes a critical interval with respect to the current edge e_i. ¿From this lemma we assume in the succeeding discussion that $C(e_i)$ is the one defined in the lemma statement.

Lemma 6. The above algorithm runs in $O(n \log n)$ time by using dynamic trees of [9].

Proof. Similarly done as in the proof of Theorem 1. □

After obtaining all pairs of $(e, upper(e))$ for each $e \in T_{min}$, the desired cut is computed as explained at the beginning of this section. This is justified by the following lemma.

Lemma 7. If $r_1 \leq r_2 - r_3$, then $C(e^*)$ is a minimum range cut. Otherwise, $C(e^{**}) \cup C(e^{***})$ is a minimum range target cut.

Proof. Suppose $r_1 \leq r_2 - r_3$ holds and $C(e^*)$ is not a minimum range target cut. Let C' be a desired minimum range target, $f_* =$ *min-edge(C')*, and $f^* =$ *max-edge(C')*. Note that $w(f_*) \leq \gamma \leq w(f^*)$. Suppose that $\gamma = f_*$ or $\gamma = f^*$. Thus, $C' \in \mathcal{C}_1$, and then $w(f^*) - w(f_*) \geq r_1$, which is a contradiction. Therefore, we assume that $w(f_*) < \gamma < w(f^*)$. Suppose now that $w(upper(f_*)) \geq \gamma$, which means that $C(f_*) \in \mathcal{C}_1$. From the upper criticality of $upper(f_*)$ and $f_*, f^* \in C'$, we have $w(upper(f_*)) \leq w(f^*)$. Therefore, we have $w(f^*) - w(f_*) \geq w(upper(f_*)) - w(f_*) \geq r_1$, which is a contradiction. Thus, $w(upper(f_*)) < \gamma$. In a similar way, we can prove that $w(lower(f^*)) > \gamma$. Therefore, $C(f_*) \in \mathcal{C}_1$ and $C(f^*) \in \mathcal{C}_2$. From the assumption, $w(f^*) - w(f_*) \geq r_1$, which is a contradiction. We can prove the other case in a similar way as above. □

The entire algorithm is constructed as follows.

Step 1: Compute T_{min} and T_{max} and let $E = T_{min} \cup T_{max}$.
Step 2: Compute an upper-critical interval with respect to e for each $e \in T_{min}$ and three sets $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$. Compute r_1, r_2 and r_3 as well as the corresponding three cuts $C(e^*)$, $C(e^{**})$ and $C(e^{***})$.
Step 3: If $r_1 \leq r_2 - r_3$, output $C(e^*)$ as a minimum range target cut. Else output $C(e^{**}) \cup C(e^{***})$.

Theorem 2. The above algorithm correctly computes a minimum range target cut in $O(MST(m, n) + n \log n)$ time. □

As in the minimum range cut problem, the time complexity shown in the above theorem becomes $O(n \log n)$ for planar or Euclidean graphs.

5 Conclusion

In this paper, we devised $O(MST(m,n) + n \log n)$ time algorithms for the minimum range cut problem and its variant. Our algorithms improved the previous $O(m^2)$ time bound, which can be obtained by Martello et al.'s general approach [6]. The following two ingredients lead to this improvement: they are, (1) we only have to process edges in a minimum or maximum spanning tree to solve range cut problems; (2) for any cut, its minimum (*resp.* maximum) weight edge belongs to a minimum (*resp.* maximum) spanning tree. We hope that these are of self interest, and are now investigating on the further applications of these observations.

Acknowledgements

This work was partially supported by Grant in Aid for Scientific Research of the Ministry of Education, Science and Culture of Japan under Grant-in-Aid for Co-operative Research (A) 02302047 (1990).

References

[1] P.M. Camerini, F. Mafioli, S. Martello, and P. Toth, Most and Least Uniform Spanning Trees. *Discrete Applied Mathematics*, Vol. 15, 181-197. 1986.

[2] D. Cheriton and R.E. Tarjan, Finding Minimum Spanning Trees. *SIAM Journal on Computing*, Vol. 5, 724-742. 1976.

[3] M.L. Fredman and D.E. Willard, Trans-Dichotomous Algorithms for Minimum Spanning Trees and Shortest Paths. *Proceedings of the IEEE 31st Annual Symposium on Foundations of Computer Science*, 719-725. 1990.

[4] H.N. Gabow, Z. Galil, T. Spencer, and R.E. Tarjan, Efficient Algorithms for Finding Minimum Spanning Trees in Undirected and Directed Graphs. *Combinatorica*, Vol. 6, No. 2, 109-122. 1986.

[5] Z. Galil and B. Schieber, On Finding Most Uniform Spanning Trees. *Discrete Applied Mathematics*, Vol. 20, 173-175. 1987.

[6] S. Martello, W.R. Pulleyblank, P. Toth, and D. de Werra, Balanced Optimization Problems. *Operations Research Letters*, Vol. 3, No. 5, 275-278. 1984.

[7] C. Monma, M. Paterson, S. Suri, and F. Yao, Computing Euclidean Maximum Spanning Trees. *Proceedings of the Fourth Annual ACM Symposium on Computational Geometry*, 241-251. 1988.

[8] F.P. Preparata and M.I. Shamos, *Computational Geometry*, Springer Verlag, New York, NY. 1985.

[9] D.D. Sleator and R.E. Tarjan, A Data Structure for dynamic Trees. *Journal of Computer and System Sciences*, Vol. 26, 362-391. 1983.

[10] R.E. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA. 1983.