Vol.2016-AL-158 No.12
2016/6/25

最大重み極小セパレータ問題について

土中 哲秀[1,a)]　Bodlaender, Hans L. [2]　Zanden, T.C. van der[2]　小野 廣隆[1,b)]

概要：無向連結グラフ $G = (V, E)$ において, 2 点 s, t が与えられたとき, s と t を分離する点部分集合を s-t セパレータという. 特に, 任意の真部分集合が s-t セパレータではないものを極小 s-t セパレータという. 本研究では, 点重み付きグラフに対して, 重み最大の極小 s-t セパレータを発見する問題について考察する. まずこの問題が NP 困難であることを示した上で, 木幅が $\mathbf{tw}$ 以下のグラフに対して, $O^*(9^{\mathbf{tw}} \cdot W)$-時間で重み W の極小 s-t セパレータが存在するかどうかを判定する乱択アルゴリズムを提案する.

On the Maximum Weight Minimal Separator

Tesshu Hanaka[1,a)]　Hans L. Bodlaender[2]　T.C. van der Zanden[2]　Hirotaka Ono[1,b)]

Abstract: Given an undirected and connected graph $G = (V, E)$ and two vertices $s, t \in V$, a vertex subset S which separates s and t is called s-t separator. Additionally, S is called minimal s-t separator if no proper set of S separates s and t. In this pater, we consider finding a minimal s-t separator with maximum weight on a vertex-weighted graph. We firstly prove this problem is NP-hard. Then we propose an $O^*(9^{\mathbf{tw}} \cdot W)$-randomized algorithm to determine whether there exists a minimal s-t separator with weight W on a graph with bounded treewidth.

1. Introduction

Given a connected graph $G = (V, E)$ and two vertices $s, t \in V$, a set $S \subseteq V$ of vertices is called an *s-t separator* if s and t belong to different connected components in $G \setminus S$, where $G \setminus S = (V \setminus S, E)$. If a set S is an s-t separator for some s and t, it is simply called a *separator*. If an s-t separator S is minimal in terms of set inclusion, that is, no proper subset of S separates s and t, it is called a *minimal s-t separator*. Similarly, if a separator is minimal in terms of set inclusion, it is called a *minimal separator*.

Separators and minimal separators have been considered important in several contexts and have been intensively studied indeed. For example, they are obviously related to the connectivity of graphs, which is an important notion in many practical applications, such as network design, supply chain analysis and so on. From a theoretical point of view, minimal separators are related to treewidth or potential maximal cliques, which play key roles in designing fast algorithms [3], [4].

In this paper, we consider the problem of finding the most important minimal separator of a given weighted graph. More precisely, the problem is defined as follows: Given a connected graph $G = (V, E)$, vertices $s, t \in V$ and a weight function $w : V \to \mathbb{N}^+$, find a minimal s-t separator whose weight $\sum_{v \in S} w(v)$ is maximum. The decision version of the problem is to decide the existence of minimal s-t separator with weight W. We name the problems Maximum Weight Minimal s-t Separator.

This problem is motivated in the context of supply chain network analysis. When a weighted network represents a supply chain where a vertex represents an industry, s and t are virtual vertices respectively represents source and sink, and its weight of a vertex represents its financial importance, the maximum weight minimal s-t separator is interpreted as the set of industries that is most significant

[1] Department of Economic Engineering, Kyushu University
[2] Department of Computing Science Algorithmic Systems, Utrecht University
[a)] 3EC15004S@s.kyushu-u.ac.jp
[b)] hirotaka@econ.kyushu-u.ac.jp

but vulnerable in the supply chain network.

Unfortunately, the problem is shown to be NP-hard, and we then design an FPT algorithm with respect to treewidth. It should be noted that since the condition of s-t connectivity can be written with Monadic Second Order Logic, it can be solved in $f(\mathbf{tw})\cdot n$ time by Courcelle's meta-theorem, where f is a computable function and $\mathbf{tw}$ is treewidth. However, the function f forms a tower of exponentials; the existence of an FPT algorithm with better running time is not obvious. In this paper, we design an $O^*(c^{\mathbf{tw}} \cdot W)$-time randomized algorithm for the decision version, where c is a constant and O^* is the order notation omitting the polynomial factor, by utilizing the technique called *Cut & Count*; the running time is bounded by a single exponential of treewidth. Furthermore, by applying the technique of convolution technique, we improve the running time by reducing the base of the exponent from $c = 21$ to $c = 9$; the running time of the resulting algorithm is $O^*(9^{\mathbf{tw}} \cdot W)$.

1.1 Related work

Minimal separators have been investigated long time from many aspects. As mentioned above, they are related to treewidth or potential maximal cliques, for example [3], [4]. In general, a graph has exponentially many minimal separators, and in fact there exists a graph with $\Omega(3^{n/3})$ minimal separators [7]. On the other hand, some graphs have only polynomially (even linearly) many minimal separators. For example, a chordal bipartite graph has polynomial number of separators [10]. For such a class of graphs, Maximum Weight Minimal s-t Separator can be solved in polynomial time, because we just evaluate the weights after enumerating all the separators.

The remainder of the paper is organized as follows. In Section 2, we first give basic terminology and basic notions of designing algorithms. Then we show NP-hardness of our problem in Section 3. Finally, we design randomized algorithms based on *Cut & Count* technique.

2. Preliminaries

In this section, we describe some notations and definitions. Let $G = (V, E)$ be an undirected and vertex-weighted graph. For $V' \subseteq V$, let $G[V']$ denote a subgraph of G induced by V'. Furthermore, we denote the set of neighbors of v by $N(v)$ for a vertex v. We define the function $[p]$ such that if p is true, it returns 1, otherwise 0.

2.1 Tree Decomposition

Our algorithms that will be presented in Sections 3 and 4 are based on dynamic programming on tree decomposition. In this subsection, we give the definition of tree decomposition.

Definition 2.1. *A* tree decomposition *of a graph $G = (V, E)$ is defined as a pair $\langle \mathcal{X}, T\rangle$, where $\mathcal{X} = \{X_1, X_2, \ldots, X_N \subseteq V\}$, and T is a tree whose nodes are labeled by $1, 2, \ldots, N$, such that*

1. $\bigcup_{i\in I} X_i = V$.
2. *For $\forall\{u, v\} \in E$, there exists X_i such that $\{u, v\} \subseteq X_i$.*
3. *For all $i, j, k \in \{1, 2, \ldots, N\}$, if j lies on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.*

In the following, we call T a decomposition tree, and we use term "nodes" (not "vertices") for T to avoid a confusion. Moreover, we call a subset of V corresponding to a node $i \in I$ a *bag* and denote it by X_i. The width of a tree decomposition $\langle \mathcal{X}, T\rangle$ is defined by $\max_{i\in\{1,2,\ldots,N\}} |X_i|-1$, and the treewidth of G, denoted by $\mathbf{tw}(G)$, is the minimum width over all tree decompositions of G. We sometimes use the notation $\mathbf{tw}$ instead of $\mathbf{tw}(G)$ by simplicity.

In general, computing $\mathbf{tw}(G)$ of a given G is NP-hard [1], but fixed-parameter tractable with respect to itself [2]. In the following, we assume that a decomposition tree with the minimum treewidth is given.

Then, Kloks introduced a very useful tree decomposition for some algorithms, called *nice tree decomposition*[9]. In the sense, it is a special binary tree decomposition which has four types of nodes, named *leaf* , *introduce vertex* , *forget* and *join*. Moreover, Cygan et al. added a new type of node named *introduce edge* [5]. This tree decomposition is called *new nice tree decomposition*.

Definition 2.2. *A tree decomposition $\langle \mathcal{X}, T\rangle$ is called* new nice tree decomposition *if it satisfies the following:*

1. *T is rooted at a designated node $X_N \in \mathcal{X}$, called root node.*
2. *Every node of the tree T has at most two children nodes.*
3. *The nodes of T hold one of the following five node types:*
   - *A* leaf *node i which has no children and the corresponding leaf bag X_i has $|X_i| = 0$.*
   - *An* introduce vertex *node i which has one child j with $X_i = X_j \cup \{v\}$ for a vertex $v \in V$.*
   - *An* introduce edge *node i which has one children j and labeled with an edge $(u, v) \in E$ where $u, v \in X_i = X_j$.*

- *A* Forget *node i which has one child j with $X_i = X_j \setminus \{v\}$ for a vertex $v \in V$.*
- *A* Join *node i which has two children nodes $j, l \in \mathcal{X}$ with $X_i = X_j = X_l$.*

We can transform any tree decompositions to new nice tree decompositions in polynomial time. Here, given a tree decomposition $\langle \mathcal{X}, T \rangle$, we define a subgraph $G_t = (V_t, E_t)$ for each node t where V_t is the set of vertices added in each introduce vertex node and E_t is the set of edges added in each introduce edge node until node t on a decomposition tree.

2.2 Path Decomposition

A *path decomposition* is a tree decomposition which is the path, in other words, each node has only one children node. The width of path decomposition is defined like tree decomposition, that is, $\max_{i \in \{1,2,\ldots,N\}} |X_i| - 1$. The *path-width* is defined as minimum width of path decomposition of G and denoted by **pw**.

2.3 Isolation Lemma

In this subsection, we explain the Isolation lemma introduced by Mulmuley et al.[11]. In *Cut & Count* technique, it is used for obtaining a single solution with high probability. Therefore, the Isolation lemma allows us to count objects modulo 2.

Definition 2.3 ([11]). *A function $w : U \to \mathbb{Z}$ isolates a set family $\mathcal{F} \subseteq 2^U$ if there is a unique $S' \in 2^F$ with $w(S') = \min_{S \in \mathcal{F}} w(S)$ where $w(X) = \sum_{u \in X} w(u)$.*

Lemma 2.1 (Isolation Lemma[11]). *Let $F \subseteq 2^U$ be a set family over a universe U with $|F| > 0$. For each $u \in U$, choose a weight $w(u) \in \{1, 2, \ldots N\}$ uniformly and independently at random. Then*

$$Pr[w \text{ isolate } \mathcal{F}] \geq 1 - \frac{|U|}{N}.$$

2.4 Cut & Count

The *Cut & Count* technique was introduced by Cygan et al. for solving connectivity problems[5]. The concept of *Cut & Count* is counting the number of relaxed solutions such that we do not consider whether they are connected or disconnected. Then we compute the number of relaxed solutions modulo 2 and we determine whether there exists a connected solution by cancellation tricks. Now, we define a *consistent cut* to explain the detail of *Cut& Count*.

Definition 2.4 ([5]). *A cut (V_1, V_2) of $V' \subseteq V$ such that $V_1 \cup V_2 = V'$ and $V_1 \cap V_2 = \emptyset$ is* consistent *if $v_1 \in V_1$ and $v_2 \in V_2$ implies $(v_1, v_2) \notin E$.*

This means that consistent cut (V_1, V_2) of V' has no edge between V_1 and V_2. By the definition of a consistent cut, we can explain the *Cut& Count*.

Let $\mathcal{S} \subseteq 2^U$ be a set of solutions. According to [5] and [6], *Cut & Count* is divided into two parts as follows.

- **The Cut part** : Relax the connectivity requirement by considering the set $\mathcal{R} \supseteq \mathcal{S}$ of possibly connected or disconnected candidate solutions. Moreover, consider the set $\mathcal{C}$ of pairs $(X; C)$ where$X \in \mathcal{R}$ and C is a consistent cut of X.
- **The Count part** : Isolate a single solution by sampling weights of all elements in U with high probability by the isolation lemma. Then, compute $|\mathcal{C}|$ modulo 2 using a sub-procedure. Disconnected candidate solutions $X \in \mathcal{R} \setminus \mathcal{S}$ cancel since they are consistent with an even number of cuts. If the only connected candidate $x \in \mathcal{S}$ exists, we obtain the odd number of cuts .

Given a set U and a tree decomposition $\langle \mathcal{X}, T \rangle$, the general scheme of *Cut & Count* is as follows:

Step 1. Set the weight for every vertex uniformly and independently at random by $w : U \to \{1, \ldots, 2|U|\}$.

Step 2. For each weight $0 \leq W \leq 2|U|^2$, compute the number of relaxed solutions of weight W with consistent cuts on a decomposition tree.

Step 3. In the root node, return yes if it is odd. otherwise no.

Therefore, we design the concrete function that computes the number of relaxed solutions of Maximum weighted s-t minimal separator in Section 4.

3. NP-hardness

In this section, we show that Maximum Weight Minimal s-t Separator is NP-hard.

Theorem 3.1. Maximum Weight Minimal s-t Separator *is strongly NP-hard.*

Proof. We show the reduction from MAX CUT of unweighted graph $G = (V, E)$, which asks the existence of cut $(C, V \setminus C)$ whose value $|\{(u, v) \in E \mid u \in C, v \in V \setminus C\}|$ is at least k. This problem is shown to be NP-hard in [8].

In the following, we construct an instance of Maximum Weight Minimal s-t Separator from $G = (V, E)$ and positive integer k. Let $p = (3n+1)k$ and $G' = (V \cup E \cup V' \cup V'' \cup \{s, t\}, E_1 \cup E_2)$, where $n = |V|$, $V' = \{v' \mid v \in V\}$, $V'' = \{v'' \mid v \in V\}$, $E_1 = \bigcup_{e=(u,v) \in E} \{(u, e), (v, e)\}$ and $E_2 = \bigcup_{u \in V} \{(s, u'), (u', u)\} \cup \bigcup_{u \in V} \{(t, u''), (u'', u)\}$. The vertex weights of G' are defined by $w_v = 3n + 1$ for $v \in E$

and 1 otherwise.

We first show that if G has a cut C of weight at least k, G' has a minimal s-t separator S whose weight is at least $p = (3n+1)k$. We define $S = \{u'' \in V'' \mid u \in V \cap C\} \cup \{v' \in V' \mid v \in V \setminus C\} \cup \{e = (u,v) \in E \mid u \in C, v \in V \setminus C\}$. The weight of S is at least $(3n+1)k$ by $|\{e = (u,v) \in E \mid u \in C, v \in V \setminus C\}| \geq k$. We can see that S is a s-t separator, since any $u \in C$ is reachable from s even after removing S from G' but not reachable from t. The minimality also holds because by adding any vertex in S we obtain a path from s to t.

We next show that if G' has a minimal s-t separator S whose weight is at least $p = (3n+1)k$, G has a cut C of weight at least k. By the weighting, S contains at least k vertices in E. Note that for any $v \in V(G)$, at least one of v, v', v'' is included in S, otherwise S does not separate s and t. If S does not contain any $v \in V$, let C be vertices in V that are reachable from s after removing S; C is actually a cut, and its weight is k. Otherwise, S contains a vertex $v \in V$. In this case, S does not contain any e forming $e = (v,x)$ because otherwise it contradicts the minimality. Then, we construct $S' := S \setminus \{v\} \cup \{v'\} \cup \{e = (u,x) \in E\}$. By repeating this procedure, we obtain S that does not contain any $v \in V$.

This completes the proof. □

4. Algorithms using *Cut & Count*

In this section, we design algorithms on path/tree decomposition using *Cut & Count* technique.

4.1 Bounded pathwidth

We give an algorithm that solves Maximum Weight Minimal s-t Separator in time $O^*(9^{\mathbf{pw}} \cdot W)$ for graphs of pathwidth at most **pw**. This algorithm is based on *Cut & Count* technique.

Definition 4.1. *A relaxed solution of weight W is a partition (S, A, B, Q) of V, so that $\Sigma_{v \in S} w(v) = W$, $s \in A, t \in B$ and for all $v \in S$, there exist vertices $a \in A, b \in B$ so that $(a,v) \in E, (v,b) \in E$ and for A, B, Q, there does not exist an edge (u, v) such that u and v are in different sets.*

Theorem 4.1. *There exists a minimal s-t separator of weight W if and only if there exists a relaxed solution (S, A, B, Q) of weight W so that A and B are connected.*

Proof. ($\Rightarrow$) Let S be a minimal s-t separator of weight W. Let A be the connected component of $V \setminus S$ containing s and similarly, B be the connected component containing t and let $Q = V \setminus A \setminus B \setminus S$. Note that $S \cap A \cap B \cap Q = \emptyset$ and there is no edge between A, B and Q. We claim that (S, A, B, Q) is a relaxed solution of weight W. We consider a vertex $v \in S$. Suppose that $v \in S$ has no edge (a,v) for any vertices $a \in A$. Since S is minimal s-t separator, $G[V \setminus (S \setminus \{v\}]$ has s-t path. However, for A, B, Q, there is no edge between each other and $(a,v) \notin E$. Thus, $G[V \setminus (S \setminus \{v\}]$ does not have s-t path. It contradicts that S is minimal. Hence, there exists vertex $a \in A$ such that $(a,v) \in E$ for all vertices $v \in S$. Similarly, there exists vertex $a \in B$ such that $(v,b) \in E$ for all vertices $v \in S$. Note that A and B are connected (by construction), $s \in A$ and $t \in B$ and that the relaxed solution is clearly of weight W.

($\Leftarrow$) Suppose that (S, A, B, Q) is a relaxed solution of weight W so that A and B are connected. We claim that S is a minimal separator. Suppose (for the purpose of contradiction) that there exists a vertex $v \in S$ so that $S \setminus \{v\}$ separates s and t. Then there exist vertices $a \in A, b \in B$ so that $(a,v) \in E, (v,b) \in E$. Since A is connected, there exist a path (in A) from s to a. Similarly, there exist a path (in B) from b to t. Joining these paths with the edges (a,v) and (v,b) gives a path from s to t, contradicting that $S \setminus \{v\}$ is a separator. Hence S is a minimal separator, and by definition it is of weight W. □

To determine whether a relaxed solution (S, A, B, Q) of weight W so that A and B are connected exists, we use a variation on the *Cut & Count* technique. Since we require connectedness of both A and B, we consider consistent cuts of A and B.

Definition 4.2. *Given a node t of the tree decomposition of G, a* partial solution *for that node is a tuple $(S_\emptyset, S_A, S_B, S_{AB}, A_l, A_r, B_l, B_r, Q, w)$, so that:*

- $V_t = S_\emptyset \cup S_A \cup S_A \cup S_{AB} \cup A_l \cup A_r \cup B_l \cup B_r \cup Q$
- (A_l, A_r) *is a consistent cut: there exists no edge $(u,v) \in E$ such that $u \in A_l$ and $v \in A_r$.*
- (B_l, B_r) *is a consistent cut: there exists no edge $(u,v) \in E$ such that $u \in B_l$ and $v \in B_r$.*
- $w = \Sigma_{v \in S} w(v)$
- *For all $v \in S_\emptyset$, $N(v) \cap (A_l \cup A_r \cup B_l \cup B_r) = \emptyset$*
- *For all $v \in S_A$, $N(v) \cap (B_l \cup B_r) = \emptyset$ and $N(v) \cap (A_l \cup A_r) \neq \emptyset$*
- *For all $v \in S_B$, $N(v) \cap (B_l \cup B_r) \neq \emptyset$ and $N(v) \cap (A_l \cup A_r) = \emptyset$*
- *For all $v \in S_B$, $N(v) \cap (B_l \cup B_r) \neq \emptyset$ and $N(v) \cap (A_l \cup A_r) \neq \emptyset$*

Here, we set another weight $w'(v)$ for each vertex by choosing from $\{1, \ldots, 2|V|\}$ uniformly and independently

at random for the Isolation lemma. We also set the coloring $c : V \to \{s_\emptyset, s_A, s_B, s_{AB}, a_l, a_r, b_l, b_r, q\}$. Especially, the coloring of vertex v is denoted by $c(v)$. They represent the state of vertex v, for example, v is in $S_\emptyset$ if $c(v) = s_\emptyset$. Then we give a dynamic programming algorithm that computes the number of partial solutions.

To count the number of relaxed solutions with consistent cuts, for each coloring c, w and w' we define the function $f_x(c, w, w')$ as follows.

Leaf node:

In a leaf node, we define $f_x(\emptyset, 0, 0) = 1$, if $S_\emptyset = S_A = S_B = S_{AB} = A_l = A_r = B_l = B_r = \emptyset$ and $w, w' = 0$. Otherwise, $f_x(c, w, w') = 0$.

Introduce vertex v node:

The function f_x has five cases in introduce vertex nodes. Note that we only add one vertex v without edges, thus if $c(v) = s_A, s_B, s_{AB}$, it is not a solution.

$$f_x(c \times \{c(v)\}, w, w') :=$$

$$\begin{cases} [v \neq s, t] f_y(c, w - w(v), w' - w'(v)) & (c(v) = s_\emptyset) \\ [v \neq t] f_y(c, w, w') & (c(v) = a_l) \\ [v \neq s] f_y(c, w, w') & (c(v) = b_l) \\ [v \neq s, t] f_y(c, w, w') & (c(v) = a_r, b_r, q) \\ 0 & (c(v) = s_A, s_B, s_{AB}). \end{cases}$$

Introduce edge (u, v) node:

For introduce edge nodes, we check each state of endpoint of edge (u, v).

$c(u) = s_\emptyset \Rightarrow$

$$\begin{aligned} & f_x(c \times \{c(u)\} \times \{c(v)\}, w, w') \\ & := [c(v) \neq a_l, a_r, b_l, b_r] f_y(c \times \{s_\emptyset\} \times \{c(v)\}, w, w') \end{aligned}$$

$c(u) = s_A \Rightarrow$

$$\begin{aligned} & f_x(c \times \{c(u)\} \times \{c(v)\}, w, w') \\ & := [c(v) = a_l, a_r] f_y(c \times \{s_\emptyset\} \times \{c(v)\}, w, w') \\ & + [c(v) \neq b_l, b_r] f_y(c \times \{s_A\} \times \{c(v)\}, w, w') \end{aligned}$$

$c(u) = s_B \Rightarrow$

$$\begin{aligned} & f_x(c \times \{c(u)\} \times \{c(v)\}, w, w') \\ & := [c(v) = b_l, b_r] f_y(c \times \{s_\emptyset\} \times \{c(v)\}, w, w') \\ & + [c(v) \neq a_l, a_r] f_y(c \times \{s_B\} \times \{c(v)\}, w, w') \end{aligned}$$

$c(u) = s_{AB} \Rightarrow$

$$\begin{aligned} & f_x(c \times \{c(u)\} \times \{c(v)\}, w, w') \\ & := [c(v) = b_l, b_r] f_y(c \times \{s_A\} \times \{c(v)\}, w, w') \\ & + [c(v) = a_l, a_r] f_y(c \times \{s_B\} \times \{c(v)\}, w, w') \\ & + f_y(c \times \{s_{AB}\} \times \{c(v)\}, w, w') \end{aligned}$$

$c(u) = a_l, a_r, b_l, b_r, q \Rightarrow$

$$\begin{aligned} & f_x(c \times \{c(u)\} \times \{c(v)\}, w, w') \\ & := [c(v) = c(u)] f_y(c \times \{c(u)\} \times \{c(v)\}, w, w') \end{aligned}$$

Forget v node:

For forget nodes, the state of v will never change forward. Thus, if $c(v) = s_\emptyset, s_A, s_B$, its separator is not minimal. For this reason, we only sum up for each state $c(v) = s_{AB}, a_l, a_r, b_l, b_r, q$. The function f_x in a forget node is as follows:

$$\begin{aligned} & f_x(c, w, w') \\ & := \sum_{c(v) = s_{AB}, a_l, a_r, b_l, b_r, q} f_y(c \times \{c(v)\}, w, w'). \end{aligned}$$

Therefore, the following theorem holds.

Theorem 4.2. *For graphs of pathwidth at most* **pw***, there exists a Monte-Carlo algorithm that solves* Maximum Weight Minimal s-t Separator *in time* $O^*(9^{\mathbf{pw}} \cdot W)$

4.2 Bounded treewidth

Next, we show that given graphs of treewidth at most **tw**, there exists a Monte-Carlo algorithm that solves Maximum Weight Minimal s-t Separator in time $O^*(21^{\mathbf{tw}} \cdot W)$ by defining the function for join bags.

Join node:

We suppose join node x has two children nodes y, z and denote each coloring by c_x, c_y, c_z and weights of solution sets in x, y, z by $x_x, w_y, w_z, w'_x, w'_y, w'_z$. Suppose that if $c_x(v) = a_l, a_r, b_l, b_r, q$, then $c_y(v) = c_z(v) = c_x(v)$. We also denote two subsets of bags corresponding to nodes y, z by T_y, T_z, respectively. Note that $T_y \subseteq X_y$ and $T_z \subseteq X_z$. For simplicity, we denote the vertex subset $c^{-1}(\{s_\emptyset, s_A, s_B, s_{AB}\}) \subseteq X_x = X_y = X_z$ by S^*.

Then we sum up all combinations of vertex states. For each vertex, there exist twenty-one combinations. The function f_x is defined as follows:

$$
\begin{aligned}
&f_x(c_x, w_x, w'_x)\\
&:= \sum_{w_y+w_z=w_x+w(S^*)} \sum_{w'_y+w'_z=w'_x+w'(S^*)} \sum_{T_y,T_z\subseteq S^*}\\
&[\{T_y \cup T_z = c_x^{-1}(\{s_\emptyset, s_A, s_B, s_{AB}\})\} \wedge\\
&\{\forall v \in c_x^{-1}(\{s_{AB}\}), c_y(v) = s_{AB} \wedge c_z(v) = s_\emptyset, s_A, s_B)\\
&\vee(c_y(v) = s_\emptyset, s_A, s_B \wedge c_z(v) = s_{AB})\\
&\vee(c_y(v) = s_A \wedge c_z(v) = s_B)\\
&\vee(c_y(v) = s_B \wedge c_z(v) = s_A)\}\\
&\wedge\{\forall v \in c_x^{-1}(\{s_A\}), (c_y(v) = s_A \wedge c_z(v) = s_\emptyset)\\
&\vee(c_y(v) = s_\emptyset \wedge c_z(v) = s_A)\\
&\vee(c_y(v) = s_A \wedge c_z(v) = s_A)\}\\
&\wedge\{\forall v \in c_x^{-1}(\{s_B\}), (c_y(v) = s_B \wedge c_z(v) = s_\emptyset)\\
&\vee(c_y(v) = s_\emptyset \wedge c_z(v) = s_B)\\
&\vee(c_y(v) = s_B \wedge c_z(v) = s_B)\}\\
&\wedge\{\forall v \in c_x^{-1}(\{s_\emptyset\}), (c_y(v) = s_\emptyset \wedge c_z(v) = s_\emptyset)]\\
&f_y(c_y, w_y, w'_y) f_z(c_z, w_z, w'_z).
\end{aligned}
$$

Theorem 4.3. *For graphs of treewidth at most* $\mathbf{tw}$*, there exists a Monte-Carlo algorithm that solves* Maximum Weight Minimal s-t Separator *in time* $O^*(21^{\mathbf{tw}} \cdot W)$

Moreover, we show that there exists a Monte-Carlo algorithm that solves Maximum Weight Minimal s-t Separator in time $O^*(9^{\mathbf{tw}} \cdot W)$ using the convolution technique[12].

First, we set the new coloring $\hat{c} : V \to \{s_{\bar{A}\bar{B}}, s_{\bar{A}}, s_{\bar{B}}, s_{all}, a_l, a_r, b_l, b_r, q\}$. The state $s_{\bar{A}\bar{B}}$ represents that a vertex v is in S and has no neighbor of A and B. The state $s_{\bar{A}}(s_{\bar{B}})$ represents a vertex v is in S and has no neighbor of $A(B)$, respectively. Finally, the state s_{all} represents a vertex v is in S without constraints.

Then, we show the following lemma to transform between c and $\hat{c}$.

Lemma 4.1. *Let* x *be a node of a tree decomposition and* $f_x(c, w, w')$ *be a counting function to count the number of partial solutions of* Maximum Weighted Minimal s-t Separator *of each weight* w, w'*, corresponding to each coloring* c*,* $\hat{c}$ *of a node* x*. Then we can transform from one coloring to another coloring for each function without loss of information. In addition, it is computed in* $O(W \cdot W' \cdot 9^{\mathbf{tw}} \cdot |X_x|)$.

Proof. This proof follows [12]. We consider an immediate step i in transformation. For $f_x(c \times \{c(v)\} \times \hat{c}, w, w')$, v is a vertex which turns into the state of another coloring in i-th step and c is subcoloring of size $i-1$ and $\hat{c}$ is also subcoloring of size $|X_x|-i$. Here, for simplicity, we denote $f_x(c \times \{c(v)\} \times \hat{c}, w, w')$ and $f_x(c \times \{\hat{c}(v)\} \times \hat{c}, w, w')$ by $f_x(c(v))$ and $f_x(\hat{c}(v))$.

Transformation from c to $\hat{c}$ of f_x in i-th step is as follows:

- $f_x(s_{\bar{A}\bar{B}}) = f_x(s_\emptyset)$
- $f_x(s_{\bar{A}}) = f_x(s_\emptyset) + f_x(s_B)$
- $f_x(s_{\bar{B}}) = f_x(s_\emptyset) + f_x(s_A)$
- $f_x(s_{all}) = f_x(s_\emptyset) + f_x(s_A) + f_x(s_B) + f_x(s_{AB})$.

Conversely, we can transform from $\hat{c}$ to c as follows:

- $f_x(s_\emptyset) = f_x(s_{\bar{A}\bar{B}})$
- $f_x(s_A) = f_x(s_{\bar{B}}) - f_x(s_{\bar{A}\bar{B}})$
- $f_x(s_B) = f_x(s_{\bar{A}}) - f_x(s_{\bar{A}\bar{B}})$
- $f_x(s_{AB}) = f_x(s_{all}) - f_x(s_{\bar{A}}) - f_x(s_{\bar{B}}) + f_x(s_{\bar{A}\bar{B}})$.

Each transformation can be performed in $O(|X_x|)$-time, hence total running time of each direction is $O(W \cdot W' \cdot 9^{\mathbf{tw}} \cdot |X_x|)$-time. □

Therefore, we firstly transform from the original coloring to the new coloring in $O(W \cdot W' \cdot 9^{\mathbf{tw}} \cdot |X_x|)$-time. Then we compute the following function f_x for the new coloring $\hat{c}$:

$$
\begin{aligned}
&f_x(\hat{c}, w_x, w'_x) :=\\
&\sum_{w_y+w_z=w_x+w(\hat{S}^*)} \sum_{w'_y+w'_z=w'_x+w'(\hat{S}^*)} f_y(\hat{c}, w_y, w'_y) f_z(\hat{c}, w_z, w'_z)
\end{aligned}
$$

where $\hat{S}^* = \hat{c}^{-1}(\{s_\emptyset, s_A, s_B, s_{AB}\}) \subseteq V$. Finally, we transform the coloring conversely.

Thus, we can show the following theorem.

Theorem 4.4. *For graphs of treewidth at most* $\mathbf{tw}$*, there exists a Monte-Carlo algorithm that solves* Maximum Weight Minimal s-t Separator *in time* $O^*(9^{\mathbf{tw}} \cdot W)$.

参考文献

[1] S. Arnborg, D. G. Corneil, A. Proskurowski : Complexity of finding embeddings in a k-tree. In: *SIAM Journal on Algebraic Discrete Methods* 8(2), 277–284 (1987)

[2] H.L. Bodlaender : A linear-time algorithm for finding tree-decompositions of small treewidth. In: *SIAM Journal on Computing* 25(6), 1305–1317 (1996)

[3] H. L. Bodlaender, T. Kloks, D. Kratsch : Treewidth and pathwidth of permutation graphs. In: *SIAM Journal on Discrete Mathematics* 8(4), 606–616 (1995)

[4] V. Bouchitté, , I. Todinca : Listing all potential maximal cliques of a graph. In: *Theoretical Computer Science* 276(1-2), 17–32(2002)

[5] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M, van Rooij, J. O. Wojtaszczyk : Solving connectivity problems parameterized by treewidth in single exponential time. In: *Proceeding FOCS '11 Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science* 150–159 (2011)

[6] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk and S. Saurabh : Parameterized Algorithms, Springer (2015)

[7] F. V. Fomin, D. Kratsch, I. Todinca, Y. Villanger : Exact algorithms for treewidth and minimum fill-in. In: *SIAM Journal on Computing*, 38(3), 1058–1079 (2008).
[8] M. R. Garey, D. S. Johnson, and L. Stockmeyer : Some simplified NP-complete graph problems. In: *Theoretical computer science*, 1(3), 237–267 (1976).
[9] T. Kloks : Treewidth, Computations and Approximations. *Lecture Notes in Computer Science* 842, Springer (1994).
[10] T. Kloks, D. Kratsch : Treewidth of chordal bipartite graphs. In: *Journal of Algorithms* 19(2), 266–281 (1995)
[11] K. Mulmuley, U. V. Vazirani, V. V. Vazirani. Matching is as easy as matrix inversion. In: *Combinatorica* 7(1), 105–113 (1987)
[12] J. M. M. van Rooij, H. L. Bodlaender, P. Rossmanith : Dynamic programming on tree decompositions using generalised fast subset convolution. In: *17th Annual European Symposium on Algorithms, ESA 2009, Lecture Notes in Computer Science* 5757, Springer, 566–577 (2009)