On the maximum weight minimal separator

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

On the maximum weight minimal separator ☆,☆☆

Tesshu Hanaka [a,*], Hans L. Bodlaender [b,c], Tom C. van der Zanden [b],
Hirotaka Ono [d]

[a] *Department of Information and System Engineering, Faculty of Science and Engineering, Chuo University, Tokyo, Japan*
[b] *Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands*
[c] *Department of Mathematics and Computer Science, Eindhoven University of Technology, PO Box 513, 5600 MB Eindhoven, the Netherlands*
[d] *Department of Mathematical Informatics, Nagoya University, Nagoya, Japan*

A R T I C L E   I N F O

A B S T R A C T

Given an undirected and connected graph $G = (V, E)$ and two vertices $s, t \in V$, a vertex subset S that separates s and t is called an s-t separator, and an s-t separator is called *minimal* if no proper subset of S separates s and t. Moreover, we say that a set S is a *minimal separator* of G if S is a minimal s-t separator for some s and t. In this paper, we consider finding a minimal (s-t) separator with maximum weight on a vertex-weighted graph. We first prove that these problems are NP-hard. On the other hand, we give an $O^*(\mathbf{tw}^{O(\mathbf{tw})})$-time deterministic algorithm based on tree decompositions where O^* is the order notation omitting the polynomial factor of n. Moreover, we improve the algorithm by using the Rank-Based approach and the running time is $O^*(38 \cdot 2^\omega)^{\mathbf{tw}}$. Finally, we give an $O^*(9^{\mathbf{tw}} \cdot W^2)$-time randomized algorithm to determine whether there exists a minimal (s-t) separator where W is its weight and $\mathbf{tw}$ is the treewidth of G.

© 2019 Elsevier B.V. All rights reserved.

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 also separates s and t, it is called a *minimal s-t separator*. We say that a set S is a *minimal separator* of G if S is a minimal *s-t* separator for some s and t.

Separators and minimal separators are important in several contexts and have indeed been studied intensively. For example, they are 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 [6–8,12].

In this paper, we consider the problem of finding a maximum weight minimal *s-t* 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 call this problem MAXIMUM WEIGHT MINIMAL *s-t* SEPARATOR. Similarly, MAXIMUM WEIGHT MINIMAL SEPARATOR is the following problem: Given a connected graph $G = (V, E)$ and a weight function $w : V \to \mathbb{N}^+$, find a minimal separator S whose weight $\sum_{v \in S} w(v)$ is maximum. The decision version of the problem is to decide the existence of minimal separator with weight W.

This problem arises 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 representing source and sink respectively, and the weight of a vertex represents its financial importance, the maximum weight minimal *s-t* separator is interpreted as the most important set of industries that is influential or vulnerable in the supply chain network.

On the negative side, we show that these problems are NP-hard on bipartite graphs, even if all the vertex weights are identical, that is, the problem is to find the maximum size of minimal (*s-t*) separator. On the other hand, we show that MAXIMUM WEIGHT MINIMAL SEPARATOR is fixed-parameter tractable with respect to the solution size and weight. We then design FPT algorithms with respect to treewidth. It should be noted that since the condition of *s-t* connectivity can be written in 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 of the graph. 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 propose two parameterized algorithms for MAXIMUM WEIGHT *s-t* MINIMAL SEPARATOR with respect to treewidth. We first propose a $2^{O(\mathbf{tw} \log \mathbf{tw})} n$-time deterministic algorithm based on a standard dynamic programming approach. However, this algorithm is not a single exponential for treewidth. Thus, we use recent techniques and obtain two $O^*(c^{\mathbf{tw}})$-time algorithms, where c is a constant and O^* is the order notation omitting the polynomial factor of n. One is an $(38 \cdot 2^\omega)^{\mathbf{tw}} \mathbf{tw}^{O(1)} n$-time deterministic algorithm where ω is the matrix multiplication constant and the other is an $O^*(9^{\mathbf{tw}} \cdot W^2)$-time randomized algorithm for the decision version. The former algorithm is based on the Rank-Based approach proposed by Bodlaender et al. [4], whereas the latter is based on Cut & Count introduced by Cygan et al. [10]. These techniques are proposed to get a single exponential algorithm for connectivity problems. For the latter algorithm, by applying another technique called *fast convolution*, we improve the running time by reducing the base of the exponent from $c = 21$ to $c = 9$; the total running time of the resulting algorithm is $O^*(9^{\mathbf{tw}} \cdot W^2)$, which can be further improved when the graph is unweighted. Moreover, MAXIMUM WEIGHT MINIMAL SEPARATOR can be solved by solving $O(n^2)$ instances of the *s-t* variant (one for each combination of s and t), thus, we obtain $(38 \cdot 2^\omega)^{\mathbf{tw}} \mathbf{tw}^{O(1)} n^3$-time deterministic and $O^*(9^{\mathbf{tw}} \cdot W^2)$-time randomized algorithms.

1.1. Related work

The number of minimal separators. Minimal separators have been investigated for a long time in many aspects. As mentioned above, they are related to treewidth or potential maximal cliques, see for example [7,8,12]. In general, a graph may have exponentially many minimal separators, and in fact there exist graphs with $\Omega(3^{n/3})$ minimal separators [12]. Recently, this bound was improved to $\Omega(1.4457^n)$ [14]. On the other hand, some graph classes have only polynomially (even linearly) many minimal separators. For example, Bouchitté showed that weakly triangulated (weakly chordal) graphs have a polynomial number of separators [7]. As examples of other graph classes with polynomially many minimal separators, there are circular-arc graphs [18] and polygon-circle graphs, which are a superclass of circle graphs [22].

On the other hand, Berry et al. presented an $O(n^3 \cdot R_{\mathrm{sep}})$-time algorithm that enumerates all the minimal separators where R_{sep} is the number of these [2]. By combining these results, we know that MAXIMUM WEIGHT MINIMAL *s-t* SEPARATOR can be solved in polynomial time for the graph classes mentioned above. That is, we just enumerate all the minimal separators and evaluate the weights of these for such graphs.

Proposition 1.1. MAXIMUM WEIGHT MINIMAL *s-t* SEPARATOR *and* MAXIMUM WEIGHT MINIMAL SEPARATOR *can be solved in polynomial time for a graph classes that have a polynomial number of minimal separators.*

The relationship between minimal separators and treewidth. Minimal separators and treewidth are strongly related. As for the number of minimal separators, if a graph has a polynomially many minimal separators, we can compute its treewidth in polynomial time [7,8]. Such graph classes include (amongst others) circular-arc graphs ($O(n^2)$ [18]), polygon-circle graphs ($O(n^2)$ [22]), weakly triangulated graphs ($O(n^2)$ [7]). Furthermore, computing treewidth is fixed-parameter tractable with respect to the maximum size of a minimal separator [21]. This parameter corresponds to the solution size of MAXIMUM WEIGHT MINIMAL SEPARATOR on unweighted graphs. In this sense, this paper focuses on the converse relation of these two parameters: maximum size of a minimal separator and treewidth. That is, for treewidth as the parameter, we consider the fixed parameter tractability of MAXIMUM WEIGHT MINIMAL SEPARATOR.

The comparison with MAX CUT. The MAX CUT problem is a classical graph problem where, given an undirected and edge-weighted graph $G = (V, E)$, we have to find a set $S \subseteq V$ that maximizes $\sum_{u \in S, v \in V \setminus S} w_{uv}$, where w_{uv} is the weight of edge (u, v). Both MAX CUT and MAXIMUM WEIGHT MINIMAL SEPARATOR are in some sense connectivity problems. However, in the former problem the value of a solution is based on edge weights, whereas in the latter problem it is given by vertex weights. As such, the problems can behave quite differently: It is known that MAX CUT is NP-hard on chordal graphs [5], but

on bipartite graphs, it is trivial. In contrast, MAXIMUM WEIGHT MINIMAL SEPARATOR is NP-hard on bipartite graphs but can be solved in polynomial time on chordal graphs.

The remainder of this paper is organized as follows: In Section 2, we give basic terminology and definitions. In Section 3, we show that the problem is NP-hard, while it is fixed-parameter tractable with respect to the solution size. In Section 4, we design a basic dynamic programming algorithm based on tree decompositions and then improve the algorithm by using the Rank-Based approach. In Section 5, we give a faster randomized algorithm, using the Cut & Count technique. Finally, we conclude the paper in Section 6.

2. Preliminaries

In this section, we give notations, definitions, and some basic concepts. Let $G = (V, E)$ be an undirected, connected, and vertex-weighted graph where $|V| = n$ and $|E| = m$. When we focus on MAXIMUM WEIGHT MINIMAL s-t SEPARATOR, we assume that G does not have the edge (s, t), that is, $(s, t) \notin E$ because otherwise, there is no s-t separator. For $V' \subseteq V$, let $G[V']$ denote the subgraph of G induced by V'. Furthermore, we denote the set of neighbors of a vertex v by $N(v)$. We define the function $[p]$ as follows: if p is true, then $[p] = 1$, otherwise $[p] = 0$.

2.1. Tree decompositions

The algorithms proposed in Sections 4 and 5 are based on dynamic programming on tree decompositions. 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 T is a tree with node set I and $\mathcal{X} = \{X_i \mid i \in I\}$ is a collection of subsets of V such that:

1. $\bigcup_{i \in I} X_i = V$.
2. For every $(u, v) \in E$, there exists an $i \in I$ such that $\{u, v\} \subseteq X_i$.
3. For every $i, j, k \in I$, 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 the elements of T to avoid 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 $\max_{i \in I} |X_i| - 1$, 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 $\mathbf{tw}(G)$ for simplicity.

In general, computing $\mathbf{tw}(G)$ of a given graph G is NP-hard [1], but fixed-parameter tractable with respect to itself and there exists a linear time algorithm if treewidth is fixed [3]. In the following, we assume that a tree decomposition of minimum width is given.

Kloks introduced a very useful type of tree decomposition, called *nice tree decomposition* [17]. More precisely, it is a binary tree decomposition which has four types of nodes: *leaf*, *introduce vertex*, *forget* and *join*. A variant of the notion, using a new type of node named *introduce edge*, was introduced by Cygan et al. [11]. Using nice tree decompositions greatly simplifies the algorithms.

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

1. T is rooted at a designated node r in T satisfying $|X_r| = 0$, called the *root node*.
2. Every node of the tree T has at most two children.
3. Each node i in T has one of the following five types:
   - A *leaf* node i has no children and its bag X_i satisfies $|X_i| = 0$,
   - An *introduce vertex* node i has one child j with $X_i = X_j \cup \{v\}$ for a vertex $v \in V$,
   - An *introduce edge* node i has one child j and labeled with an edge $(u, v) \in E$ where $u, v \in X_i$ and $X_i = X_j$,
   - A *forget* node i has one child j and satisfies $X_i = X_j \setminus \{v\}$ for a vertex $v \in V$, and
   - A *join* node i has two child nodes j_1, j_2 and satisfies $X_{j_1} = X_i$ and $X_{j_2} = X_i$.

We can transform any tree decomposition to a nice tree decomposition with $O(n)$ bags and the same width in linear time [10]. Given a tree decomposition $\langle \mathcal{X}, T \rangle$, for each node i, we define a subgraph $G_i = (V_i, E_i)$, where V_i is the union of all bags X_j such that $j = i$ or j is a descendant of i in T, and $E_i \subseteq E$ is the set of all edges introduced in i (if i is an introduce edge node) and descendants of i.

3. Basic results

In this section, we give two basic results for MAXIMUM WEIGHT MINIMAL SEPARATOR. On the negative side, we show that MAXIMUM WEIGHT MINIMAL SEPARATOR and MAXIMUM WEIGHT MINIMAL s-t SEPARATOR are NP-hard. On the other hand, we show that it is fixed-parameter tractable with respect to W.
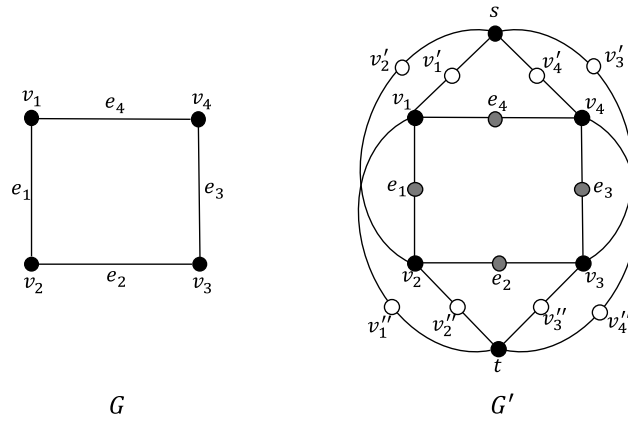
Fig. 1. An example of a construction of G' from G. Black vertices correspond to vertices in G, gray vertices correspond to edges in G, and white vertices are in V' or V''.

3.1. NP-hardness

Theorem 3.1. MAXIMUM WEIGHT MINIMAL s-t SEPARATOR *is NP-hard on bipartite graphs even if all the vertex weights are identical.*

Proof. We give a reduction from a well-known NP-hard problem, MAX CUT ([13]), which, given an unweighted graph $G = (V, E)$, asks whether there exists a cut $(C, V \setminus C)$ whose value ($|\{(u, v) \in E \mid u \in C, v \in V \setminus C\}|$) is at least k.

We construct an instance (G', p) of MAXIMUM WEIGHT MINIMAL s-t SEPARATOR from $G = (V, E)$ and positive integer k. We first provide a proof for the weighted problem and then explain how to get rid of the weights.

We build the vertex set of G' by taking the union of three copies (V, V', V'') of the vertex set V of G, where we write v' (resp., v'') to denote the vertex corresponding to the copy of $v \in V$ in V' (resp., V''). Furthermore, for each edge e in the edge set E of G, we create a corresponding vertex e in G'. Finally, we also create two vertices s and t.

We build the edge set of G' by making every vertex of V' adjacent to s, and every vertex of V'' adjacent to t. We make every vertex $v \in V$ adjacent to its corresponding copies v' (in V') and v'' (in V''). Finally, for each edge vertex e (corresponding to some edge $e = (u, v)$ in G), we take the edges (e, u) and (e, v).

Formally, let $G' = (V \cup E \cup V' \cup V'' \cup \{s, t\}, E_1 \cup E_2)$, where $V' = \{v' \mid v \in V\}$, $V'' = \{v'' \mid v \in V\}$, $E_1 = \bigcup_{e \in E}\{(u, e), (v, e) \mid e = (u, v)\}$ 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 to be $w_v = 3n + 1$ if $v \in E$ and 1 otherwise (where $n = |V|$). The graph G' can easily seen to be bipartite, by partitioning the vertices into sets $\{s, t\} \cup V$ and $V' \cup V'' \cup E$. We give an example of a construction of G' from G in Fig. 1.

We now show that if G has a cut C of weight at least k, then G' has a minimal s-t separator S whose weight is at least $p = (3n + 1)k$.

Given C, we construct S by taking the union of the vertices in V'' that correspond to some vertex in C, the vertices in V' that correspond to some vertex not in C, and the vertices corresponding to edges bisected by C. Formally, let $S = \{u'' \in V'' \mid u \in V \cap C\} \cup \{v' \in V' \mid v \in V \setminus C\} \cup \{(u, v) \in E \mid u \in C, v \in V \setminus C\}$.

S is a s-t separator: it is easy to see that the vertices reachable from s (after removing S) are precisely the vertices in V and V' that correspond to vertices in C, together with vertices corresponding to edges between vertices in C. On the other hand, the vertices reachable from t are precisely the vertices in V and V'' that correspond to vertices *not* in C, together with vertices corresponding to edges between vertices *not* in C.

Furthermore, S is minimal, since removing from S any vertex e corresponding to an edge $e = (u, v)$ gives rise to an s-t path s, u', u, e, v, v'', t (if $u \in C, v \notin C$) or s, v', v, e, u, u'', t (if $u \notin C, v \in C$). Removing from S any vertex v' or v'' gives rise to an s-t path s, v', v, v'', t.

The weight of S is at least $(3n + 1)k$ since $|\{(u, v) \in E \mid u \in C, v \in V \setminus C\}| \geq k$ (i.e., since C is a cut that bisects at least k edges, our separator S includes at least k edge vertices).

We next show that if G' has a minimal s-t separator S whose weight is at least $p = (3n + 1)k$, then 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 $S' := S \setminus \{v\} \cup \{v'\} \cup \{(v, x) \in E\}$ - obtaining a minimal separator of greater weight. By repeating this procedure, we obtain a minimal separator S of weight at least $(3n + 1)k$ that does not contain any $v \in V$. This completes the correctness of the reduction.

As mentioned above, this reduction can be modified to the unweighted case. To this end, we create $3n + 1$ identical copies of each edge vertex e (and of its incident edges). In the new reduction, to block the path between u and v, we need to remove $3n + 1$ copies of e ($= (u, v)$), which plays the same role as the original vertex having weight $3n + 1$. □

Next, we show how to adapt this proof to the case of MAXIMUM WEIGHT MINIMAL SEPARATOR, which does not require the separator to separate s and t (but rather only requires that the separator separates *some* pair of vertices).

Corollary 3.2. MAXIMUM WEIGHT MINIMAL SEPARATOR *is NP-hard on bipartite graphs, even if all the vertex weights are identical.*

Proof. We give a reduction from MAXIMUM WEIGHT MINIMAL s-t SEPARATOR. Given an instance (G, p, s, t, w) of MAXIMUM WEIGHT MINIMAL s-t SEPARATOR, we add an additional vertex x which we make adjacent to both s and t and we give x weight $w(x) = \Sigma_{v \in V} w(v) + 1$. We ask whether there exists a minimal separator of weight at least $w(x) + p$. If S is a separator of weight at least $w(x) + p$, it must necessarily include x, and thus, be an s-t separator. It then follows that $S \setminus \{x\}$ must be an s-t separator of weight at least p in G. The converse easily follows: if S is an s-t separator in G of weight at least p, then $S \cup \{x\}$ is a separator of weight at least $p + w(x)$ in the modified graph.

The proof for the unweighted case follows similarly to above, by creating $w(x)$ identical copies of x (and noting that our hardness proof for MAXIMUM WEIGHT MINIMAL s-t SEPARATOR uses polynomial weights). □

On the other hand, we show that MAXIMUM WEIGHT MINIMAL SEPARATOR is fixed-parameter tractable with respect to the solution size k.

Theorem 3.3. *For unweighted graphs G,* MAXIMUM WEIGHT MINIMAL SEPARATOR *is fixed-parameter tractable with respect to the solution size k.*

Proof. We first determine whether G contains $k \times k$ grid minor or not in $f(k) \cdot n^2$-time [16]. If G has an $k \times k$ grid minor, then G also has a minimal separator of size at least k. Otherwise, the treewidth of G is at most $g(k)$ by the excluded grid theorem [20,9]. Therefore, we can use dynamic programming on a tree decomposition of width bounded by a function of k. As the MAXIMUM WEIGHT MINIMAL SEPARATOR problem can be formulated in Monadic Second Order Logic, we can solve the problem in linear time for fixed k. In this paper, we give more efficient dynamic programming algorithms; see Section 4. □

For vertex-weighted graphs, MAXIMUM WEIGHT MINIMAL SEPARATOR is also fixed-parameter tractable with respect to W.

Corollary 3.4. *For vertex-weighted graphs G,* MAXIMUM WEIGHT MINIMAL SEPARATOR *is fixed-parameter tractable with respect to W.*

4. Dynamic programming on tree decompositions

In this section we present an FPT algorithm for MAXIMUM WEIGHT MINIMAL s-t SEPARATOR, parameterized by treewidth. We first present an algorithm running in time $\mathbf{tw}^{O(\mathbf{tw})}n$. We then show how to improve this algorithm using the Rank-Based approach to get a single-exponential time algorithm.

4.1. A $\mathbf{tw}^{O(\mathbf{tw})}n$-time algorithm

To be able to use "standard" algorithmic techniques, we reformulate the MAXIMUM WEIGHT MINIMAL s-t SEPARATOR as a connectivity problem. In doing so, we can use techniques such as the Rank-Based approach and Cut & Count [11,10]. We start with defining the notion of connected s-t partition:

Definition 4.1. A *connected s-t partition* of weight W is a partition (S, A, B, Q) of V such that: (1) $s \in A, t \in B$, (2) $G[A]$ is connected, (3) $G[B]$ is connected, (4) $\sum_{v \in S} w(v) = W$, (5) for $\forall v \in S$, there exist vertices $a \in A, b \in B$ such that $(a, v) \in E, (v, b) \in E$ and (6) for sets A, B, Q, there does not exist an edge (u, v) such that u and v are in different sets.

Note that S of (S, A, B, Q) is an s-t separator due to Definition 4.1 (6). The key to the design of our algorithms is the following lemma, which states that connected s-t partitions correspond to minimal s-t separators.

Theorem 4.2. *There exists a minimal s-t separator of weight W if and only if there exists a connected s-t partition (S, A, B, Q) of weight W.*

To prove Theorem 4.2, we use the following lemma. This lemma appears in many papers and books, for example, as an exercise in [15].

Lemma 4.3 ([15]). *Let S be a minimal s-t separator and A, B be the connected components of $G[V \setminus S]$ containing s and t, respectively. Then every vertex of S has a neighbor in A and a neighbor in B.*
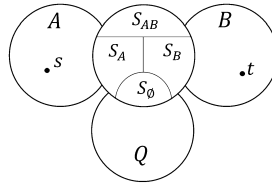
Using this lemma, we can prove Theorem 4.2.

Fig. 2. Connection between vertex sets.

Theorem 4.2. ($\Rightarrow$) Let S be a minimal s-t separator of weight W. Let A be the subset of vertices of $V \setminus S$ such that $G[A]$ is the connected component containing s and B be the subset of vertices such that $G[B]$ is the connected component containing t. Moreover, let $Q = ((V \setminus A) \setminus B) \setminus S$. Note that S, A, B and Q are pairwise disjoint and that there is no edge between A, B and Q. By Lemma 4.3, all vertices in S have neighbors in both A and B. Therefore, (S, A, B, Q) is a connected s-t partition of weight W.

($\Leftarrow$) Suppose that (S, A, B, Q) is a connected s-t partition of weight W. We claim that S is a minimal s-t separator. To show this by contradiction, suppose that S of partition (S, A, B, Q) is an s-t separator, which is not minimal; there exists a vertex $v \in S$ such that $S \setminus \{v\}$ separates s and t. Due to Definition 4.1 (5), there exist vertices $a \in A, b \in B$ such that $(a, v) \in E, (v, b) \in E$. Since $G[A]$ is connected, there exists a path (in $G[A]$) from s to a. Similarly, there exists a path (in $G[B]$) from b to t. By joining these paths with the edges (a, v) and (v, b), we obtain a path from s to t, which contradicts that $S \setminus \{v\}$ is a separator. Hence S is a minimal s-t separator, and by definition it is of weight W. $\square$

Using connected s-t partitions, we design a $\mathbf{tw}^{O(\mathbf{tw})}n$-time algorithm for Maximum Weight Minimal s-t Separator. First, we partition S into $S_\emptyset$, S_A, S_B and S_{AB} (see Fig. 2). Set $S_\emptyset$ consists of the vertices in S that have no neighbor in A and B, but may have neighbors in S_A, S_B, S_{AB} or Q. Set S_A (resp., S_B) consists of the vertices in S that have at least one neighbor in A (resp., B), but no neighbor in B (resp., A). They may have neighbors in S_A, S_B, S_{AB}, Q. Set S_{AB} consists of the vertices in S that have neighbors in A and in B and may have neighbors in S_A, S_B, S_{AB}, Q.

Since we eventually want the sets A and B to become connected, we need to track their connectivity. We define two partitions $\mathcal{P}^A = \{P_1^A, P_2^A, \ldots, P_\alpha^A\}$ of $X_i \cap A$ and $\mathcal{P}^B = \{P_1^B, P_2^B, \ldots, P_\beta^B\}$ of $X_i \cap B$. We call each element of a partition P_ℓ^A (resp., P_ℓ^B) a *block*. They correspond to the intersection of X_i and the vertex sets of connected components of $G_i^A = (A \cap V_i, E_i)$ (resp. $G_i^B = (B \cap V_i, E_i)$). Note that there are at most $|X_i|^{O(|X_i|)}$ partitions for each node X_i. α and β are the number of connected components in $G_i^A = (A \cap V_i, E_i)$ and $G_i^B = (B \cap V_i, E_i)$, respectively. Note that $\alpha \leq |X_i \cap A|$ and $\beta \leq |X_i \cap B|$. Intuitively, one block $\{\{v\}\}$ is added to $\mathcal{P}^A$ in each introduce vertex v node; then blocks may be merged in introduce edge nodes and join nodes. In a forget node, note that a vertex may not be the last of its block, else, the solution is invalid and must be discarded since its component of A is not connected.

Combining these a tree decomposition $\langle \mathcal{X}, T \rangle$, we will define a *partial solution*. For each node i, let $G_i = (V_i, E_i)$ be a subgraph defined as right after Definition 2.2. The definition of *partial solution* is as follows.

Definition 4.4. Given a node i of the tree decomposition $\langle \mathcal{X}, T \rangle$ of G, a *partial solution* for node i is a partition $(S_\emptyset, S_A, S_B, S_{AB}, A, B, Q)$ of V_i with $\mathcal{P}^A$ and $\mathcal{P}^B$, such that:

- $S_\emptyset \cup S_A \cup S_A \cup S_{AB} \cup A \cup B \cup Q = V_i$,
- $\forall v \in S_\emptyset, N(v) \cap (A \cup B) = \emptyset$,
- $\forall v \in S_A, N(v) \cap B = \emptyset$ and $N(v) \cap A \neq \emptyset$,
- $\forall v \in S_B, N(v) \cap B \neq \emptyset$ and $N(v) \cap A = \emptyset$,
- $\forall v \in S_{AB}, N(v) \cap B \neq \emptyset$ and $N(v) \cap A \neq \emptyset$,
- $\forall v_1, v_2 \in A, v_1, v_2$ are in the same block in $\mathcal{P}^A \leftrightarrow v_1, v_2$ are connected in G_i^A,
- $\forall v_1, v_2 \in B, v_1, v_2$ are in the same block in $\mathcal{P}^B \leftrightarrow v_1, v_2$ are connected in G_i^B,
- $s \in V_i \Rightarrow s \in A$, and
- $t \in V_i \Rightarrow t \in B$.

Let $\Sigma = \{s_\emptyset, s_A, s_B, s_{AB}, a, b, q\}$. Each element of Σ is called *state* of vertices (e.g., we say that vertex $v \in S_\emptyset$ has state $s_\emptyset$). Then, we define the *coloring* function $c : V \to \Sigma$. The coloring function represents which set of the partition a vertex is in, for example, if v is in $S_\emptyset$ then $c(v) = s_\emptyset$. For $\Sigma' \subseteq \Sigma$, we also define $c^{-1}(\Sigma') \subseteq V$ which represents the set of vertices whose states are in Σ'.

Given two sets V and W, we denote their colorings by $c_V \in \Sigma^{|V|}$ and $c_W \in \Sigma^{|W|}$, respectively. Suppose that V and W are disjoint, $c_V = (c(v_1), \ldots, c(v_{|V|}))$, and $c_W = (c(w_1), \ldots, c(w_{|W|}))$, where $v_i \in V$ and $w_i \in W$. We then define the *concatenation* $c_V \times c_W \in \Sigma^{|V|+|W|}$ of c_V and c_W as the coloring $(c(v_1), \ldots, c(v_{|V|}), c(w_1), \ldots, c(w_{|W|}))$. Moreover, if $W \subseteq V$, then we define the *separation* $c_V \setminus c_W \in \Sigma^{|V|-|W|}$ as the coloring $(c(v_{i_1}), \ldots, c(v_{i_{|V|-|W|}}))$, where $v_{i_1}, \ldots, v_{i_{|V|-|W|}} \in V \setminus W$.

In our algorithm, we assume we are given a nice tree decomposition of minimum width. We transform this tree decomposition by adding $\{s, t\}$ to all bags; thus we can suppose that the root bag X_r contains exactly two vertices s and t. The width of this tree decomposition is at most $\mathbf{tw} + 2$.

We define the function $f_i(c, \mathcal{P}^A, \mathcal{P}^B)$ to be the possible maximum weight of vertices in $S \cap V_i$ of a partial solution $(S_\emptyset, S_A, S_B, S_{AB}, A, B, Q)$ of V_i with $\mathcal{P}_A$ and $\mathcal{P}_B$. If $c, \mathcal{P}^A, \mathcal{P}^B$ deviate from the definition of a partial solution, then let $f_i(c, \mathcal{P}^A, \mathcal{P}^B) = -\infty$.

We now give recursive formulas for computing f_i in each node i. In the root node, $f_r(\{c(s)\} \times \{c(t)\}, \{\{s\}\}, \{\{t\}\}) = f_r(\{a\} \times \{b\}, \{\{s\}\}, \{\{t\}\})$ is the maximum weight of minimal s-t separators because $X_r = \{s, t\}$.

In the following, we let i denote a parent node, and let j denote its corresponding child node. For a join node, we write j_1 and j_2 to denote its two children. To emphasize that we are dealing with two different colorings, we denote parent node colorings by c_i and child node colorings by c_j.

Leaf node: In leaf nodes, if $c(s) = a$ and $c(t) = b$, we define $f_i(\{c(s)\} \times \{c(t)\}, \{\{s\}\}, \{\{t\}\}) := 0$. Otherwise, $f_i(\{c(s)\} \times \{c(t)\}, \mathcal{P}^A, \mathcal{P}^B) := -\infty$ since there are only two vertices s, t in X_i.

Introduce vertex v node: In introduce vertex nodes, we consider three cases for colorings. If $c(v) = s_\emptyset$, we add $w(v)$ to $f_j(c, \mathcal{P}^A, \mathcal{P}^B)$ because v is added in S. If $c(v) \in \{a, b, q\}$, the value of f_i does not change since $v \notin S$. Moreover, we add a block $\{\{v\}\}$ to $\mathcal{P}^A$ or $\mathcal{P}^B$ depending on whether $c(v) = a$ or $c(v) = b$, respectively. Finally, if $c(v) \in \{s_A, s_B, s_{AB}\}$, a partial solution is invalid by the definition because v has no incident edge and hence no neighbor in A or B. Therefore, we define f_i as follows:

$$f_i(c, \mathcal{P}^A, \mathcal{P}^B) := \begin{cases} f_j(c \setminus \{c(v)\}, \mathcal{P}^A, \mathcal{P}^B) + w(v) & \text{if } c(v) = s_\emptyset \\ f_j(c \setminus \{c(v)\}, \mathcal{P}^A \setminus \{\{v\}\}, \mathcal{P}^B) & \text{if } c(v) = a \\ f_j(c \setminus \{c(v)\}, \mathcal{P}^A, \mathcal{P}^B \setminus \{\{v\}\}) & \text{if } c(v) = b \\ f_j(c \setminus \{c(v)\}, \mathcal{P}^A, \mathcal{P}^B) & \text{if } c(v) = q \\ -\infty & \text{otherwise.} \end{cases}$$

Introduce edge (u, v) node: In introduce edge nodes, we define f_i for the following cases of $c(u), c(v)$.

- If $c(u) = a$ and $c(v) = a$, the vertices u, v are in A. If u and v are in the different blocks, we set $f_i(c, \mathcal{P}^A, \mathcal{P}^B) := -\infty$ because u and v are in the same block of partition $\mathcal{P}^A$ in node i due to edge (u, v). Then, there are two cases: the partitions in $A \cap X_i$ (parent) and $A \cap X_j$ (child) are same or not. In the former case, u and v are in the same block in the partition of $A \cap X_j$, and we then set $f_i(c, \mathcal{P}^A, \mathcal{P}^B) := f_j(c, \mathcal{P}^A, \mathcal{P}^B)$. In the latter case, let $\mathcal{P}'^A$ be a partition of $A \cap X_j$ such that $\mathcal{P}^A \neq \mathcal{P}'^A$ but $\mathcal{P}'^A$ changes to $\mathcal{P}^A$ by merging two blocks of $\mathcal{P}'^A$ including u and v respectively with edge (u, v). Therefore, we take a $\mathcal{P}'^A$ that maximizes $f_i(c, \mathcal{P}'^A, \mathcal{P}^B)$. Then, we set f_i as follows:

$$f_i(c, \mathcal{P}^A, \mathcal{P}^B) := \max\{f_j(c, \mathcal{P}^A, \mathcal{P}^B), \max_{\mathcal{P}'^A} f_j(c, \mathcal{P}'^A, \mathcal{P}^B)\}.$$

- The case that $c(u) = b$ and $c(v) = b$ is almost the same as the case that $c(u) = a$ and $c(v) = a$. If u and v are not in the same block of partition $\mathcal{P}^B$, we then set $f_i(c, \mathcal{P}^A, \mathcal{P}^B) := -\infty$. Let $\mathcal{P}'^B$ be a partition of $B \cap X_j$ such that $\mathcal{P}^B \neq \mathcal{P}'^B$ but $\mathcal{P}'^B$ changes to $\mathcal{P}^B$ by merging two blocks of $\mathcal{P}'^B$ including u and v respectively with edge (u, v). Then, we define f_i as follows:

$$f_i(c, \mathcal{P}^A, \mathcal{P}^B) := \max\{f_j(c, \mathcal{P}^A, \mathcal{P}^B), \max_{\mathcal{P}'^B} f_j(c, \mathcal{P}^A, \mathcal{P}'^B)\}.$$

- If $c(u), c(v) \in \{s_\emptyset, s_A, s_B, s_{AB}, q\}$, we define f_i as follows:

$$f_i(c, \mathcal{P}^A, \mathcal{P}^B) = f_j(c, \mathcal{P}^A, \mathcal{P}^B).$$

  In this case, (u, v) is irrelevant to the partitions and the value is not changed because only one edge (u, v) is added.

- If $(c(u), c(v)) = (s_A, a), (a, s_A)$, we consider two cases. One case is that $u \in S_A$ and $v \in A$ in the child node and the other case is that $u \in S_\emptyset$ and $v \in A$ in the child node. In the other case, u is moved from $S_\emptyset$ into S_A by adding (u, v), because u has a neighbor v in A. Thus, we define f_i as follows:

$$f_i(c \times \{s_A\} \times \{a\}, \mathcal{P}^A, \mathcal{P}^B) := \max\{f_j(c \times \{s_A\} \times \{a\}, \mathcal{P}^A, \mathcal{P}^B), f_j(c \times \{s_\emptyset\} \times \{a\}, \mathcal{P}^A, \mathcal{P}^B)\}.$$

- If $(c(u), c(v)) = (s_B, b), (b, s_B)$, we consider almost the same cases as above; that is, $u \in S_B, v \in B$ and $u \in S_\emptyset$ and $v \in B$ in the child node.

$$f_i(c \times \{s_B\} \times \{b\}, \mathcal{P}^A, \mathcal{P}^B) := \max\{f_j(c \times \{s_B\} \times \{b\}, \mathcal{P}^A, \mathcal{P}^B), f_j(c \times \{s_\emptyset\} \times \{b\}, \mathcal{P}^A, \mathcal{P}^B)\}.$$

Table 1

This table represents combinations of states of two child nodes j_1, j_2 for each vertex in $X_i = X_{j_1} = X_{j_2}$. The rows and columns correspond to states of j_1, j_2 respectively and inner elements correspond to states of x. For example, if $c_i(v) = s_A$, there are three combinations such that $(c_{j_1}(v), c_{j_2}(v)) = (s_A, s_\emptyset)$, $(c_{j_1}(v), c_{j_2}(v)) = (s_\emptyset, s_A)$ and $(c_{j_1}(v), c_{j_2}(v)) = (s_A, s_A)$.

	$s_\emptyset$	s_A	s_B	s_{AB}	a	b	q
$s_\emptyset$	$s_\emptyset$	s_A	s_B	s_{AB}	a	b	q
s_A	s_A	s_A	s_{AB}	s_{AB}			
s_B	s_B	s_{AB}	s_B	s_{AB}			
s_{AB}	s_{AB}	s_{AB}	s_{AB}	s_{AB}			
a					a		
b						b	
q							q

- If $(c(u), c(v)) = (s_{AB}, a), (a, s_{AB})$, there are two cases: (1) $u \in S_{AB}$ and $v \in A$ in the child node and (2) $u \in S_B$ and $v \in A$ in the child node. In the latter case, u is moved from S_B to S_{AB} by adding (u, v), because u has a neighbor v in B. Therefore, we define f_i as follows:

$$f_i(c \times \{s_{AB}\} \times \{a\}, \mathcal{P}^A, \mathcal{P}^B) := \max\{f_j(c \times \{s_{AB}\} \times \{a\}, \mathcal{P}^A, \mathcal{P}^B), f_j(c \times \{s_B\} \times \{a\}, \mathcal{P}^A, \mathcal{P}^B)\}.$$

- If $(c(u), c(v)) = (s_{AB}, b), (b, s_{AB})$, we consider almost the same cases as above; that is, $u \in S_{AB}$, $v \in B$ and $u \in S_A$ and $v \in B$ in the child node.

$$f_i(c \times \{s_{AB}\} \times \{b\}, \mathcal{P}^A, \mathcal{P}^B) := \max\{f_j(c \times \{s_{AB}\} \times \{b\}, \mathcal{P}^A, \mathcal{P}^B), f_j(c \times \{s_A\} \times \{b\}, \mathcal{P}^A, \mathcal{P}^B)\}.$$

- Otherwise, we set $f_i(c, \mathcal{P}^A, \mathcal{P}^B) := -\infty$ because the rest of the cases are invalid by the definition of states and partial solution. There must not exist edge (u, v) for such cases.

Forget v node: In a forget v node, if $c_j(v) \in \{s_\emptyset, s_A, s_B\}$, vertex v will never have neighbors both in A and in B and hence this case is invalid because of the definition of the connected s-t partition, which requires that each vertex in S has neighbors in both A and B. If $c_j(v) \in \{s_{AB}, q\}$, we need not consider the connectivity of v. In the case that $c_j(v) = a$, we only consider partitions such that there exists at least one vertex u in A included in the same block as v. If there is no such vertex, the block including v is never merged. Consequently, $G[A]$ would not be connected in the root node. The case that $c_j(v) = b$ is almost the same. Let $\mathcal{P}'^A, \mathcal{P}'^B$ be a partition satisfying such conditions, then we define f_i as follows:

$$f_i(c, \mathcal{P}^A, \mathcal{P}^B) := \max\{f_j(c \times \{s_{AB}\}, \mathcal{P}^A, \mathcal{P}^B), f_j(c \times \{q\}, \mathcal{P}^A, \mathcal{P}^B),$$
$$\max_{\mathcal{P}'^A} f_j(c \times \{a\}, \mathcal{P}'^A, \mathcal{P}^B), \max_{\mathcal{P}'^B} f_j(c \times \{b\}, \mathcal{P}^A, \mathcal{P}'^B)\}.$$

Join node: For a parent node i and two child nodes j_1, j_2, we denote the corresponding colorings by c_i, c_{j_1}, c_{j_2} and the corresponding partitions by $\mathcal{P}_i^A, \mathcal{P}_i^B, \mathcal{P}_{j_1}^A, \mathcal{P}_{j_1}^B, \mathcal{P}_{j_2}^A, \mathcal{P}_{j_2}^B$. We then define a subset D of tuples of $((c_{j_1}, \mathcal{P}_{j_1}^A, \mathcal{P}_{j_1}^B), (c_{j_2}, \mathcal{P}_{j_2}^A, \mathcal{P}_{j_2}^B))$ such that the combinations of colorings for c_{j_1}, c_{j_2} satisfy the following conditions (see Table 1):

- $\forall v \in c_i^{-1}(\{s_\emptyset, a, b, q\}), (c_{j_1}(v), c_{j_2}(v)) = (c_i(v), c_i(v))$,
- $\forall v \in c_i^{-1}(\{s_A\}), (c_{j_1}(v), c_{j_2}(v)) = (s_A, s_\emptyset), (s_\emptyset, s_A), (s_A, s_A)$,
- $\forall v \in c_i^{-1}(\{s_B\}), (c_{j_1}(v), c_{j_2}(v)) = (s_B, s_\emptyset), (s_\emptyset, s_B), (s_B, s_B)$, and
- $\forall v \in c_i^{-1}(\{s_{AB}\}), (c_{j_1}(v), c_{j_2}(v)) = (s_{AB}, s_\emptyset), (s_{AB}, s_A), (s_{AB}, s_B),$
  $(s_{AB}, s_{AB}), (s_\emptyset, s_{AB}), (s_A, s_{AB}), (s_B, s_{AB}), (s_A, s_B), (s_B, s_A),$

and the partition obtained by merging $\mathcal{P}_{j_1}^A$ and $\mathcal{P}_{j_2}^A$ equals $\mathcal{P}_i^A$ and the partition obtained by merging $\mathcal{P}_{j_1}^B, \mathcal{P}_{j_2}^B$ equals $\mathcal{P}_i^B$. If $D = \emptyset$ for $c_i, \mathcal{P}_i^A, \mathcal{P}_i^B$, we set $f_i(c_i, \mathcal{P}_i^A, \mathcal{P}_i^B) := -\infty$. Otherwise, we set $S^* := c_i^{-1}(\{s_\emptyset, s_A, s_B, s_{AB}\})$. Then we define f_i as follows:

$$f_i(c_i, \mathcal{P}_i^A, \mathcal{P}_i^B) := \max_{((c_{j_1}, \mathcal{P}_{j_1}^A, \mathcal{P}_{j_1}^B), (c_{j_2}, \mathcal{P}_{j_2}^A, \mathcal{P}_{j_2}^B)) \in D} \{f_{j_1}(c_{j_1}, \mathcal{P}_{j_1}^A, \mathcal{P}_{j_1}^B) + f_{j_2}(c_{j_2}, \mathcal{P}_{j_2}^A, \mathcal{P}_{j_2}^B) - w(S^*)\}.$$

The subtraction in the right hand side of the equation above is because the weight $w(S^*)$ is counted twice; once in each child node.

We recursively calculate f_i on the decomposition tree. Note that all bags have $|X_i|$ vertices and the number of combinations of colorings and partitions $(c, \mathcal{P}^A, \mathcal{P}^B)$ in each node is $|X_i|^{O(|X_i|)} = \mathbf{tw}^{O(\mathbf{tw})}$. The running time to compute all f_i's

in X_i is dominated by join nodes and it is roughly $(\mathbf{tw}^{O(\mathbf{tw})})^3 = \mathbf{tw}^{O(\mathbf{tw})}$ since we scan every coloring and partition in two children nodes X_{j_1} and X_{j_2} for each coloring c_i and each partition $\mathcal{P}^A, \mathcal{P}^B$ and then check all combinations. Therefore, the total running time is $\mathbf{tw}^{O(\mathbf{tw})}n$ and we conclude with the following theorem.

Theorem 4.5. *For graphs of treewidth at most* **tw***, there exists an algorithm that solves* MAXIMUM WEIGHT MINIMAL s-t SEPARATOR *in time* $\mathbf{tw}^{O(\mathbf{tw})}n$.

MAXIMUM WEIGHT MINIMAL SEPARATOR can be solved by applying the above $\mathbf{tw}^{O(\mathbf{tw})}n$-time algorithm for all possible combinations of s and t, i.e., at most n^2 times. We thus obtain the following result:

Theorem 4.6. *For graphs of treewidth at most* **tw***, there exists an algorithm that solves* MAXIMUM WEIGHT MINIMAL SEPARATOR *in time* $\mathbf{tw}^{O(\mathbf{tw})}n^3$.

4.2. An improved algorithm using the Rank-Based approach

The running time of the algorithm presented in the previous section is dominated by the need to keep track of all $\mathbf{tw}^{O(\mathbf{tw})}$ possible partitions $(\mathcal{P}^A, \mathcal{P}^B)$. In this section, we show how to modify this algorithm to obtain an $O^*(c^{\mathbf{tw}})$-time algorithm using the Rank-Based approach where c is a constant. In the following, we assume that the coloring function c is fixed, and that we are dealing with fixed, disjoint sets of vertices $A, B \subseteq X_i$.

The Rank-Based approach, introduced by Bodlaender et al. [4], states that, roughly, we do not need to keep track of partial solutions for *all* possible partitions, but that it suffices to track only a *representative subset* of them.

In the following, we let $\Pi(U)$ denote the set of partitions of the set U, and for $p, q \in \Pi(U)$, let $p \sqcup q$ denote their transitive closure. A weighted set of partitions of U is a subset $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$.

Bodlaender et al. [4] say that a weighted set of partitions $\mathcal{Q}' \subseteq \Pi(U) \times \mathbb{N}$ represents another set $\mathcal{Q} \subseteq \Pi(U) \times \mathbb{N}$, if for all $p \in \Pi(U)$ it holds that:

$$\min\{w \mid (q, w) \in Q : p \sqcup q = \{U\}\} = \min\{w \mid (q, w) \in Q' : p \sqcup q = \{U\}\}$$

Theorem 4.7 *(Bodlaender et al. [4]).* *There exists an algorithm* `reduce` *that, given a set of weighted partitions* $\mathcal{Q} \subseteq \Pi(U) \times \mathbb{N}$, *outputs a set of weighted partitions* $\mathcal{Q}'$ *that represents* $\mathcal{Q}$ *of size at most* $2^{|U|-1}$, *and runs in time* $|\mathcal{Q}|2^{(\omega-1)|U|}|U|^{O(1)}$.

By performing a reduce operation after every update of the dynamic programming tables, the size of the dynamic programming tables can be kept bounded by a single-exponential function of **tw**. Doing so, Bodlaender et al. [4] obtain $O^*(c^{\mathbf{tw}})$-time algorithms for STEINER TREE, FEEDBACK VERTEX SET and HAMILTONIAN CYCLE. However, each of these problems deals with a single connectivity constraint, whereas we have two disjoint sets (A and B), for both of which we want to enforce connectivity. For this case, we need a different notion of representation:

Definition 4.8. We say that a weighted set of partitions $\mathcal{Q}' \subseteq \Pi(A \cup B) \times \mathbb{N}$ represents another set $\mathcal{Q} \subseteq \Pi(A \cup B) \times \mathbb{N}$, if for all $p \in \Pi(A \cup B)$ it holds that:

$$\max\{w \mid (q, w) \in Q : p \sqcup q = \{A, B\}\} = \max\{w \mid (q, w) \in Q' : p \sqcup q = \{A, B\}\}$$

Note that since we are dealing with a maximization problem rather than a minimization problem, we have used max in the definition rather than min. However, this difference is inconsequential.

We now show the equivalent of Theorem 4.7 for this modified notion of representation. We follow the presentation and proof of Bodlaender et al. [4] closely. We start by defining the matrix $\mathcal{M}$, analogously to Definition 3.11 of [4]:

Definition 4.9. Define $\mathcal{M} \in \mathbb{Z}_2^{\Pi(A \cup B) \times \Pi(A \cup B)}$ by

$$\mathcal{M}[p, q] = \begin{cases} 1 & p \sqcup q = \{A, B\}, \\ 0 & \text{else.} \end{cases}$$

To show that matrix $\mathcal{M}$ has bounded rank, we write it as the product of two matrices $\mathcal{C}$ and $\mathcal{C}^t$.

Definition 4.10 *(Cf. Definition 3.12 of [4]).* Let $\text{cuts} = \{(V_1, V_2) \mid V_1 \dot\cup V_2 = A \cup B \wedge s \in V_1, t \in V_2\}$. Define $\mathcal{C} \in \mathbb{Z}^{\Pi(A \cup B) \times \text{cuts}}$ by

$$\mathcal{C}[p, (V_1, V_2)] = \begin{cases} 1 & (V_1, V_2) \sqsubseteq p, \\ 0 & \text{else.} \end{cases}$$

Here $(V_1, V_2) \sqsubseteq p$ denotes that p is a refinement of (V_1, V_2), i.e., each element of p is a subset of either V_1 or V_2.

Lemma 4.11 *(Cf. Lemma 3.13 of [4]). It holds that $\mathcal{M} = \mathcal{CC}^t$.*

Proof. The matrix entry $\mathcal{CC}^t[p, q]$ is equal (modulo 2) to the number of elements of `cuts` that are consistent with both p and q; this is precisely the number of elements of `cuts` consistent with $p \sqcup q$ (i.e., the transitive closure of p and q). This number is 1 if $p \sqcup q = \{A, B\}$, and a multiple of 2 (i.e., 0 mod 2) otherwise (if s and t are in the same block of $p \sqcup q$, then no cuts are consistent with $p \sqcup q$, otherwise, if there is a block not containing p or q, it can go on either side of the cut - thus giving an even number of consistent cuts). □

Theorem 4.12. *Theorem 4.7 holds, even for representation in the sense of Definition 4.8. In fact, there exists an algorithm that, given a set of weighted partitions $\mathcal{Q} \subseteq \Pi(A \cup B) \times \mathbb{N}$, outputs a set of weighted partitions $\mathcal{Q}'$ that represents $\mathcal{Q}$ of size at most $2^{|A \cup B|-2}$, and runs in time $|\mathcal{Q}|2^{(\omega-1)|A \cup B|}|A \cup B|^{O(1)}$.*

Proof. The algorithm of Theorem 4.7 does not depend on the definition of representation; it works even if we substitute our modified definitions of $\mathcal{M}$ and $\mathcal{C}$. Note that the proof of Lemma 3.14 of [4] carries over analogously. The matrix $\mathcal{M}$ has rank bounded by $2^{|A \cup B|-2}$, since any cut (V_1, V_2) for which $s \notin V_1$ or $t \notin V_2$ is not consistent with any partition (and thus the number of non-zero rows of $\mathcal{C}$ is at most $2^{|A \cup B|-2}$). □

Theorem 4.13. *There exists an algorithm solving* Maximum Weight Minimal s-t Separator *in deterministic time $(38 \cdot 2^\omega)^{\mathbf{tw}} \mathbf{tw}^{O(1)} n$.*

Proof. We count the number of entries in the dynamic programming table at any time. There are at most $7^{\mathbf{tw}+1}$ cases for the coloring function c, and for each of these cases - applying the reduce algorithm after each step - we have to track at most $2^{\mathbf{tw}+1}$ partitions. A naive analysis would thus suggest that a join operation could generate $(7 \cdot 2)^{2(\mathbf{tw}+1)} = 196^{\mathbf{tw}+1}$ new entries. However, as illustrated in Table 1, we can only join entries that agree in what vertices get colors a, b or q, so the number of entries on which the `reduce` algorithm operates is instead bounded by $(19 \cdot 4)^{\mathbf{tw}+1} = 76^{\mathbf{tw}+1}$: if we fix the vertices that get color a, b or q, there are at most $4^{\mathbf{tw}+1}$ cases for the remaining vertices, and $2^{\mathbf{tw}+1}$ partitions. Joining the partitions (for one pair of colorings) gives $4^{\mathbf{tw}+1}$ cases, and joining this over the $4^{\mathbf{tw}+1}$ colorings gives $4^{\mathbf{tw}+1} \cdot (16 + 3)^{\mathbf{tw}+1}$ cases. Applying the reduce algorithm then takes $(76 \cdot 2^{\omega-1})^{\mathbf{tw}} \mathbf{tw}^{O(1)} = (38 \cdot 2^\omega)^{\mathbf{tw}} \mathbf{tw}^{O(1)}$ time, and this dominates the running time. □

Corollary 4.14. *There exists an algorithm solving* Maximum Weight Minimal Separator *in deterministic time $(38 \cdot 2^\omega)^{\mathbf{tw}} \mathbf{tw}^{O(1)} n^3$.*

5. Algorithm using Cut & Count

In this section, we give a Monte-Carlo algorithm that solves the decision version of Maximum Weight Minimal s-t Separator, that is, to decide whether there exists a minimal s-t separator with weight W in time $O^*(9^{\mathbf{tw}} \cdot W^2)$ for graphs of treewidth at most $\mathbf{tw}$. This algorithm is based on the Cut & Count technique.

5.1. Isolation Lemma

In this subsection, we explain the Isolation Lemma introduced by Mulmuley et al. [19]. The main idea of the Cut & Count technique is to obtain a single solution with high probability; we count modulo 2, and the Isolation Lemma guarantees the existence of such a single solution.

Definition 5.1 *([19]).* A function $w': U \to \mathbb{Z}$ *isolates* a set family $\mathcal{F} \subseteq 2^U$ if there is a unique $S' \in \mathcal{F}$ with $w'(S') = \min_{S \in \mathcal{F}} w'(S)$ where $w'(X) = \sum_{u \in X} w'(u)$.

Lemma 5.2 *(Isolation Lemma [19]).* Let $F \subseteq 2^U$ be a set family over a universe U with $|F| > 0$. For each $u \in U$, choose a weight $w'(u) \in \{1, 2, \dots N\}$ uniformly and independently at random. Then $Pr[w'$ isolates $\mathcal{F}] \geq 1 - |U|/N$.

5.2. Cut & Count

The Cut & Count technique was introduced by Cygan et al. for solving connectivity problems [11]. The concept of Cut & Count is counting the number of relaxed solutions, that is, 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 details of Cut & Count.

Definition 5.3 *([11]).* 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 a cut (V_1, V_2) of V' is consistent if there are no edges between V_1 and V_2. We fix an arbitrary vertex v in V_1. Then, if $G[V]$ has k components, then 2^{k-1} consistent cuts of V exist. If $G[V]$ is connected, then there only exists one consistent cut, namely $(V_1, V_2) = (V, \emptyset)$. Therefore, the number of consistent cuts is odd. Otherwise, if V does not induce a connected subgraph, the number of consistent cuts is a multiple of two. Therefore, we just need to compute the number of consistent cuts modulo 2 and return yes if the number of consistent cuts is odd, and return no otherwise. The Isolation Lemma allows us to guarantee (with high probability) that there exists a unique solution (and thus, that we indeed end up with an odd number of consistent cuts).

Let $\mathcal{S} \subseteq 2^U$ be a set of solutions. According to [11,10], the Cut & Count technique 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 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 integer weight for every vertex uniformly and independently at random by $w' : U \to \{1, \ldots, 2|U|\}$.
Step 2. For each integer weight $0 \le W' \le 2|U|^2$, compute the number of relaxed solutions of weight W' with consistent cuts modulo 2 on a decomposition tree. Then return yes if it is odd, otherwise no in the root node.

We use the Cut & Count technique to determine whether there exists a connected s-t partition (S, A, B, Q) of weight W so that A and B are connected. To apply the above scheme, we newly give the following definition of a *partial solution*. Note that we have to consider two consistent cuts of A and B.

Definition 5.4. Given a node i of the tree decomposition of G, a *partial solution* for that node is a tuple $(S_\emptyset, S_A, S_B, S_{AB}, A_l, A_r, B_l, B_r, Q, w)$, such that:

- $V_i = S_\emptyset \cup S_A \cup S_B \cup S_{AB} \cup A_l \cup A_r \cup B_l \cup B_r \cup Q$,
- (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 $v \in B_r$,
- $w = \Sigma_{v \in S} w(v)$,
- $\forall v \in S_\emptyset, N(v) \cap (A_l \cup A_r \cup B_l \cup B_r) = \emptyset$,
- $\forall v \in S_A, N(v) \cap (B_l \cup B_r) = \emptyset$ and $N(v) \cap (A_l \cup A_r) \neq \emptyset$,
- $\forall v \in S_B, N(v) \cap (B_l \cup B_r) \neq \emptyset$ and $N(v) \cap (A_l \cup A_r) = \emptyset$,
- $\forall v \in S_{AB}, N(v) \cap (B_l \cup B_r) \neq \emptyset$ and $N(v) \cap (A_l \cup A_r) \neq \emptyset$,
- $s \in V_i \Rightarrow s \in A_l$, and
- $t \in V_i \Rightarrow t \in B_l$.

For each vertex v, we set another weight $w'(v)$ by choosing from $\{1, \ldots, 2|V|\}$ independently at random. We also define the coloring function $c : V \to \{s_\emptyset, s_A, s_B, s_{AB}, a_l, a_r, b_l, b_r, q\}$. Now, we give a dynamic programming algorithm that computes the number of relaxed solutions with consistent cuts modulo 2. To compute that, for each c, w and w', we define the *counting* function $h_i : \{s_\emptyset, s_A, s_B, s_{AB}, a_l, a_r, b_l, b_r, q\}^{|X_i|} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ in each node i on a nice tree decomposition as follows.

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

Introduce vertex v node: The function h_i has five cases in introduce vertex nodes. Note that we only add one vertex v without edges. Thus, if $c(v) \in \{s_A, s_B, s_{AB}\}$, the partial solution is invalid by definition because v has no neighbor. If $c(v) = s_\emptyset$, vertex v is chosen as a vertex of S, and we hence add each weight $w(v), w'(v)$ to w, w', respectively. Moreover, v must not be s, t because s (resp., t) should be in A_l (resp., B_l). If not, it is not a connected s-t partition. Similarly, if $c(v) = a_l$ (resp., b_l), we check whether v is not t (resp., s). As for $c(v) \in \{a_r, b_r, q\}$, we also check whether v is neither s nor t. Therefore, we define h_i in introduce vertex nodes as follows:

$$
h_i(c, w, w') := \begin{cases}
[v \neq s, t] h_j(c \setminus \{c(v)\}, w - w(v), w' - w'(v)) & \text{if } c(v) = s_\emptyset \\
[v \neq t] h_j(c \setminus \{c(v)\}, w, w') & \text{if } c(v) = a_l \\
[v \neq s] h_j(c \setminus \{c(v)\}, w, w') & \text{if } c(v) = b_l \\
[v \neq s, t] h_j(c \setminus \{c(v)\}, w, w') & \text{if } c(v) \in \{a_r, b_r, q\} \\
0 & \text{otherwise.}
\end{cases}
$$

Introduce edge (u, v) *node:* In introduce edge nodes, we check each state of the endpoints of the edge (u, v) and define f_i for each case.

- If $c(u) = s_\emptyset$, vertex u has no neighbor in A and B. Hence, we define the function h_i in this case as follows:

$$h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') := [c(v) \notin \{a_l, a_r, b_l, b_r\}] \cdot h_j(c \times \{s_\emptyset\} \times \{c(v)\}, w, w').$$

- If $c(u) = s_A$, vertex u has neighbors in A but no neighbor in B. In this case, we have two cases. The first case is that $u \in S_\emptyset$ and $v \in A$ in the child node, because by adding edge (u, v) in the introduce edge (u, v) node, vertex u is moved from $S_\emptyset$ to S_A. The other case is that $u \in S_A$ and $v \notin B$ in the child node. If $v \in B$, vertex u is in S_{AB} in the parent node. We define h_i as follows. Note that only if $c(v) \in \{a_l, a_r\}$, we sum up two cases. If $c(v) \in \{b_l, b_r\}$, $h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') := 0$, and otherwise $h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') := h_j(c \times \{s_A\} \times \{c(v)\}, w, w')$.

$$h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') := [c(v) \in \{a_l, a_r\}]h_j(c \times \{s_\emptyset\} \times \{c(v)\}, w, w')$$
$$+ [c(v) \notin \{b_l, b_r\}]h_j(c \times \{s_A\} \times \{c(v)\}, w, w').$$

- The case that $c(u) = s_B$ is almost the same as in the above case, however, we swap the roles of A and B.

$$h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') := [c(v) \in \{b_l, b_r\}]h_j(c \times \{s_\emptyset\} \times \{c(v)\}, w, w')$$
$$+ [c(v) \notin \{a_l, a_r\}]h_j(c \times \{s_B\} \times \{c(v)\}, w, w').$$

- If $c(u) = s_{AB}$, we consider three cases: $u \in S_A$ and $v \in B$, $u \in S_B$ and $v \in A$, and $u \in S_{AB}$ and v is in an arbitrary set in the child node. For first and second case, vertex u is moved from S_A (resp., S_B) into S_{AB} by adding edge (u, v). If $u \in S_{AB}$, v is allowed to be in any set because a vertex in S_{AB} could connect to all sets. Therefore, we define f_i as follows:

$$h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') := [c(v) \in \{b_l, b_r\}]h_j(c \times \{s_A\} \times \{c(v)\}, w, w')$$
$$+ [c(v) \in \{a_l, a_r\}]h_j(c \times \{s_B\} \times \{c(v)\}, w, w')$$
$$+ h_j(c \times \{s_{AB}\} \times \{c(v)\}, w, w').$$

- If $c(u) \in \{a_l, a_r\}$, then $c(v) \notin \{b_l, b_r, q\}$ since there is no edge between A, B and Q by the definition of connected s-t partition. There is also no edge between A_l and A_r because (A_l, A_r) is a consistent cut. Therefore, if u is in A_l or A_r, then v is in the same set as u or is in one of S_A and S_{AB}. Note that because u is in A, v is not in $S_\emptyset$, S_B.

$$h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') := [c(v) = c(u)]h_j(c \times \{c(u)\} \times \{c(v)\}, w, w')$$
$$+ [c(v) \in \{s_A, s_{AB}\}]h_j(c \times \{c(u)\} \times \{c(v)\}, w, w').$$

- The case that $c(u) \in \{b_l, b_r\}$ is almost the same as in the above case, however, we replace A by B.

$$h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') := [c(v) = c(u)]h_j(c \times \{c(u)\} \times \{c(v)\}, w, w')$$
$$+ [c(v) \in \{s_B, s_{AB}\}]h_j(c \times \{c(u)\} \times \{c(v)\}, w, w').$$

- If $c(u) = q$, vertex u is in Q. Hence, v must be in $S_\emptyset, S_A, S_B, S_{AB}$, or Q because a vertex in Q has no neighbor in A and B by the definition of connected s-t partition.

$$h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') := [c(v) \in \{s_\emptyset, s_A, s_B, s_{AB}, q\}] \cdot h_j(c \times \{c(u)\} \times \{c(v)\}, w, w').$$

Forget v node: For forget nodes, if $c_j(v) \in \{s_\emptyset, s_A, s_B\}$, a partial solution does not satisfy the condition of connected s-t partitions because any $v \in S$ must have neighbors of both A and B. For this reason, we only sum up for each state $c_j(v) \in \{s_{AB}, a_l, a_r, b_l, b_r, q\}$. The function h_i in forget nodes is defined as follows:

$$h_i(c, w, w') := \sum_{c_j(v) \in \{s_{AB}, a_l, a_r, b_l, b_r, q\}} h_j(c \times \{c_j(v)\}, w, w').$$

Join node: We denote the coloring and weight for each partial solution in i, j_1, j_2 by c_i, c_{j_1}, c_{j_2} and $w_i, w_{j_1}, w_{j_2}, w'_i, w'_{j_1}, w'_{j_2}$, respectively. For a coloring c_i, we also define the subset D of tuples of (c_{j_1}, c_{j_2}) as the combinations of colorings of c_{j_1}, c_{j_2} as in Section 3 such that:

- $\forall v \in c_i^{-1}(\{s_\emptyset, a_l, a_r, b_l, b_r, q\}), (c_{j_1}(v), c_{j_2}(v)) = (c_i(v), c_i(v))$,
- $\forall v \in c_i^{-1}(\{s_A\}), (c_{j_1}(v), c_{j_2}(v)) = (s_A, s_\emptyset), (s_\emptyset, s_A), (s_A, s_A)$,
- $\forall v \in c_i^{-1}(\{s_B\}), (c_{j_1}(v), c_{j_2}(v)) = (s_B, s_\emptyset), (s_\emptyset, s_B), (s_B, s_B)$, and
- $\forall v \in c_i^{-1}(\{s_{AB}\}), (c_{j_1}(v), c_{j_2}(v)) = (s_{AB}, s_\emptyset), (s_{AB}, s_A), (s_{AB}, s_B), (s_{AB}, s_{AB}), (s_\emptyset, s_{AB}), (s_A, s_{AB}), (s_B, s_{AB}), (s_A, s_B), (s_B, s_A)$.

Let S^* be the vertex subset $c_i^{-1}(\{s_\emptyset, s_A, s_B, s_{AB}\})$. To sum up all combinations of vertex states and weights for counting, we define the function h_i. If $D = \emptyset$, we define $h_i(c_i, w_i, w'_i) := 0$. Otherwise,

$$h_i(c_i, w_i, w'_i) := \sum_{\substack{w_{j_1}+w_{j_2} \\ =w_i+w(S^*)}} \sum_{\substack{w'_{j_1}+w'_{j_2} \\ =w'_i+w'(S^*)}} \sum_{(c_{j_1}, c_{j_2}) \in D} h_{j_1}(c_{j_1}, w_{j_1}, w'_{j_1}) h_{j_2}(c_{j_2}, w_{j_2}, w'_{j_2}).$$

From now, we analyze the running time of this algorithm. In leaf, introduce vertex, introduce edge, and forget nodes, we can compute f_i for each coloring c and weight w, w' in $O(1)$-time because we only use $O(1)$-operations. Therefore, the total running time for them is $O^*(9^{\mathbf{tw}} \cdot W \cdot W')$. However, in join nodes, we sum up all weight combinations and coloring combinations satisfying some conditions. There are 21 coloring combinations for each vertex and $W \cdot W'$ weight combinations. Therefore, we compute all f_i's in a join node in time $O^*(21^{\mathbf{tw}} \cdot W^2)$. Note that by definition, $O(W'^2)$ is a polynomial factor.

Theorem 5.5. *For graphs of treewidth at most* **tw***, there exists a Monte-Carlo algorithm that solves the decision version of* Maximum Weight Minimal *s-t* Separator *in time* $O^*(21^{\mathbf{tw}} \cdot W^2)$*. It cannot give false positives and may give false negatives with probability at most* $1/2$*.*

Using the convolution technique [23], we can obtain a faster Monte-Carlo algorithm. The technique helps to speed up the computation for join nodes. 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 $s_{\bar{A}}$ (resp., $s_{\bar{B}}$) represents a vertex v is in S and has no neighbor of A (resp., 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 5.6. *Let i be a node of a tree decomposition and $h_i(c, w, w')$ be the counting function that gives for colorings c or $\hat{c}$, and weights w and w' the number of partial solutions of* Maximum Weight Minimal *s-t* Separator *that correspond to this coloring and weights. Given $h_i(c, w, w')$ for all colorings $c : V \to \{s_\emptyset, s_A, s_B, s_{AB}, a_l, a_r, b_l, b_r, q\}$, weights w between 0 and W and all weights w' between 0 and W', we can compute $h_i(\hat{c}, w, w')$ for all colorings $\hat{c} : V \to \{s_{\bar{A}\bar{B}}, s_{\bar{A}}, s_{\bar{B}}, s_{all}, a_l, a_r, b_l, b_r, q\}$, weights w between 0 and W and all weights w' between 0 and W', and vice versa. Both of these transformations do not lose any information, and can be executed in $O(W \cdot W' \cdot 9^{\mathbf{tw}} \cdot |X_i|)$ time.*

Proof. This proof scheme follows [23]. We consider the immediate ℓ-th step in the transformation from the original coloring c to the new coloring $\hat{c}$. (See Tables 2 and 3.) For $h_i(c \times \{c(v)\} \times \hat{c}, w, w')$, v is a vertex which turns into the state of another coloring in the ℓ-th step, and c is the partial coloring of size $\ell - 1$ and $\hat{c}$ is also the partial coloring of size $|X_i| - \ell$. Here, for simplicity, we denote $h_i(c \times \{c(v)\} \times \hat{c}, w, w')$ and $h_i(c \times \{\hat{c}(v)\} \times \hat{c}, w, w')$ by $h_i(c(v))$ and $h_i(\hat{c}(v))$.

Since h_i is the number of partial solutions with a consistent cut, the transformation from c to $\hat{c}$ of h_i in ℓ-th step is as follows:

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

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

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

Table 2
Combinations of the original coloring c in a join node.

	$s_\emptyset$	s_A	s_B	s_{AB}	a_l	a_r	b_l	b_r	q
$s_\emptyset$	$s_\emptyset$	s_A	s_B	s_{AB}					
s_A	s_A	s_A	s_{AB}	s_{AB}					
s_B	s_B	s_{AB}	s_B	s_{AB}					
s_{AB}	s_{AB}	s_{AB}	s_{AB}	s_{AB}					
a_r					a_r				
a_l						a_l			
b_r							b_r		
b_l								b_l	
q									q

Table 3
Combinations of the new coloring $\hat{c}$ in a join node.

	s_{all}	$s_{\bar{A}}$	$s_{\bar{B}}$	$s_{\bar{A}\bar{B}}$	a_l	a_r	b_l	b_r	q
s_{all}	s_{all}								
$s_{\bar{A}}$		$s_{\bar{A}}$							
$s_{\bar{B}}$			$s_{\bar{B}}$						
$s_{\bar{A}\bar{B}}$				$s_{\bar{A}\bar{B}}$					
a_r					a_r				
a_l						a_l			
b_r							b_r		
b_l								b_l	
q									q

These equations follow from equations of the transformation from c to $\hat{c}$.

We need for each transformation $O(|X_i|)$ steps as given above. Thus, the total running time of each transformation is $O(W \cdot W' \cdot 9^{\textbf{tw}} \cdot |X_i|)$. □

Therefore, we first transform the function with original colorings c to the function with new colorings $\hat{c}$ in $O(W \cdot W' \cdot 9^{\textbf{tw}} \cdot |X_i|)$-time. Then we compute the following function h_i for the new coloring $\hat{c}$ in $O(9^{\textbf{tw}} \cdot W^2)$-time:

$$h_i(\hat{c}, w, w_i') := \sum_{w_{j_1}+w_{j_2}=w_i+w(\hat{S}^*)} \sum_{w_{j_1}'+w_{j_2}'=w_i'+w'(\hat{S}^*)} h_{j_1}(\hat{c}, w_{j_1}, w_{j_1}') h_{j_2}(\hat{c}, w_{j_2}, w_{j_2}')$$

where $\hat{S}^* = \hat{c}^{-1}(\{s_\emptyset, s_A, s_B, s_{AB}\}) \subseteq V$. Note that $\hat{c}_i, \hat{c}_{j_1}, \hat{c}_{j_2}$ are the same coloring. Finally, we transform $\hat{c}$ to c. Thus, the total running time of this algorithm is $O^*(9^{\textbf{tw}} \cdot W^2)$.

Theorem 5.7. *For graphs of treewidth at most* **tw**, *there exists a Monte-Carlo algorithm that solves the decision version of* Maximum Weight Minimal Separator *and* Maximum Weight Minimal s-t Separator *in time* $O^*(9^{\textbf{tw}} \cdot W^2)$. *It cannot give false positives and may give false negatives with probability at most* $1/2$. *If the input graph is unweighted, the running time is* $O^*(9^{\textbf{tw}})$.

As usual for this type of algorithm, the probability of a false negative can be made arbitrarily small by repeating the algorithm.

6. Conclusion

In this paper, we studied Maximum Weight Minimal (s-t) Separator. We first showed Maximum Weight Minimal (s-t) Separator is NP-hard even on unweighted bipartite graphs. On the other hand, we designed an $O^*(\textbf{tw}^{O(\textbf{tw})})$-time deterministic algorithm. However, this algorithm is not a single exponential time algorithm. Then, we improved the algorithm by using the Rank-Based approach. The running time of the rank-based algorithm is $O^*((38 \cdot 2^\omega)^{\textbf{tw}})$. Moreover, we designed an $O^*(9^{\textbf{tw}} \cdot W^2)$-time randomized algorithm based on Cut & Count. This is better than the rank-based algorithm with respect to the base of treewidth. Finally, we mentioned the fixed-parameter tractability. In section 3, we showed Maximum Weight Minimal Separator is fixed-parameter tractable with respect to W, but Maximum Weight s-t Minimal Separator remains open.

References

[1] S. Arnborg, D.G. Corneil, A. Proskurowski, Complexity of finding embeddings in a k-tree, SIAM J. Algebraic Discrete Methods 8 (2) (1987) 277–284.
[2] A. Berry, J.P. Bordat, O. Cogis, Generating all the minimal separators of a graph, Int. J. Found. Comput. Sci. 11 (3) (2000) 397–403.
[3] H.L. Bodlaender, A linear-time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (6) (1996) 1305–1317.
[4] H.L. Bodlaender, M. Cygan, S. Kratsch, J. Nederlof, Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth, Inf. Comput. 243 (2015) 86–111.
[5] H.L. Bodlaender, K. Jansen, On the complexity of the maximum cut problem, Nord. J. Comput. 7 (1) (2000) 14–31.
[6] H.L. Bodlaender, T. Kloks, D. Kratsch, Treewidth and pathwidth of permutation graphs, SIAM J. Discrete Math. 8 (4) (1995) 606–616.
[7] V. Bouchitté, I. Todinca, Treewidth and minimum fill-in: grouping the minimal separators, SIAM J. Comput. 31 (1) (2001) 212–232.
[8] V. Bouchitté, I. Todinca, Listing all potential maximal cliques of a graph, Theor. Comput. Sci. 276 (1) (2002) 17–32.
[9] J. Chuzhoy, Z. Tan, Towards tight(er) bounds for the excluded grid theorem, in: Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, 2019, pp. 1445–1464.
[10] M. Cygan, F.V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh, Parameterized Algorithms, Springer International Publishing, 2015.
[11] 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: Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2011, 2011, pp. 150–159.
[12] F.V. Fomin, D. Kratsch, I. Todinca, Y. Villanger, Exact algorithms for treewidth and minimum fill-in, SIAM J. Comput. 38 (3) (2008) 1058–1079.
[13] M.R. Garey, D.S. Johnson, L.J. Stockmeyer, Some simplified np-complete graph problems, Theor. Comput. Sci. 1 (3) (1976) 237–267.
[14] S. Gaspers, S. Mackenzie, On the number of minimal separators in graphs, J. Graph Theory 87 (4) (2018) 653–659.
[15] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, 1980.
[16] K. Kawarabayashi, Y. Kobayashi, B. Reed, The disjoint paths problem in quadratic time, J. Comb. Theory, Ser. B 102 (2) (2012) 424–435.
[17] T. Kloks, Treewidth, Computations and Approximations, Lecture Notes in Computer Science, vol. 842, Springer-Verlag, Berlin, Heidelberg, 1994.
[18] T. Kloks, D. Kratsch, C. Wong, Minimum fill-in on circle and circular-arc graphs, J. Algorithms 28 (2) (1998) 272–289.
[19] K. Mulmuley, U.V. Vazirani, V.V. Vazirani, Matching is as easy as matrix inversion, Combinatorica 7 (1) (1987) 105–113.
[20] N. Robertson, P.D. Seymour, Graph minors. V. Excluding a planar graph, J. Comb. Theory, Ser. B 41 (1) (1986) 92–114.
[21] K. Skodinis, Efficient analysis of graphs with small minimal separators, in: Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 1999, 1999, pp. 155–166.
[22] K. Suchan, Minimal Separators in Intersection Graphs, Master's thesis, Akademia Górniczo-Hutnicza im. Stanisĺawa Staszica w Krakowie, 2003.
[23] J.M.M. van Rooij, H.L. Bodlaender, P. Rossmanith, Dynamic programming on tree decompositions using generalised fast subset convolution, in: Proceedings of the 17th Annual European Symposium on Algorithms, ESA 2009, 2009, pp. 566–577.