RAIRO-Oper. Res. 57 (2023) 2799–2818
https://doi.org/10.1051/ro/2023145

RAIRO Operations Research
www.rairo-ro.org

MAXIMUM WEIGHT t-SPARSE SET PROBLEM ON VECTOR-WEIGHTED GRAPHS

Yuquan Lin and Wensong Lin*

Abstract. Let t be a nonnegative integer and $G = (V(G), E(G))$ be a graph. For $v \in V(G)$, let $N_G(v) = \{u \in V(G) \setminus \{v\} : uv \in E(G)\}$. And for $S \subseteq V(G)$, we define $d_S(G; v) = |N_G(v) \cap S|$ for $v \in S$ and $d_S(G; v) = -1$ for $v \in V(G) \setminus S$. A subset $S \subseteq V(G)$ is called a t-sparse set of G if the maximum degree of the induced subgraph $G[S]$ does not exceed t. In particular, a 0-sparse set is precisely an independent set. A vector-weighted graph $(G, \vec{w}, t)$ is a graph G with a vector weight function $\vec{w} : V(G) \to \mathbb{R}^{t+2}$, where $\vec{w}(v) = (w(v; -1), w(v; 0), \ldots, w(v; t))$ for each $v \in V(G)$. The weight of a t-sparse set S in $(G, \vec{w}, t)$ is defined as $\vec{w}(S, G) = \sum_{v \in V(G)} w(v; d_S(G; v))$. And a t-sparse set S is a maximum weight t-sparse set of $(G, \vec{w}, t)$ if there is no t-sparse set of larger weight in $(G, \vec{w}, t)$. In this paper, we propose the maximum weight t-sparse set problem on vector-weighted graphs, which is to find a maximum weight t-sparse set of $(G, \vec{w}, t)$. We design a dynamic programming algorithm to find a maximum weight t-sparse set of an outerplane graph $(G, \vec{w}, t)$ which takes $O((t + 2)^4 n)$ time, where $n = |V(G)|$. Moreover, we give a polynomial-time algorithm for this problem on graphs with bounded treewidth.

Mathematics Subject Classification. 05C15.

Received September 27, 2022. Accepted September 9, 2023.

1. Introduction

Graphs in this paper are assumed to be finite, simple and undirected. Given a graph $G = (V(G), E(G))$, let $n = |V(G)|$ and $m = |E(G)|$. For $v \in V(G)$, let $N_G(v) = \{u \in V(G) \setminus \{v\} : uv \in E(G)\}$, $d_G(v) = |N_G(v)|$ and $\Delta(G) = \max_{v \in V(G)} d_G(v)$. Denote the distance between two vertices u and v in G by $d_G(u, v)$. For $S \subseteq V(G)$, we use $G[S]$ to denote the subgraph of G induced by S.

Let t be a nonnegative integer. A subset of vertices $S \subseteq V(G)$ is said to be a *t-sparse set* [12] of G if $\Delta(G[S]) \leq t$. In particular, a 0-sparse set is precisely an independent set. A t-sparse set is also called a *co-$(t + 1)$-plex* [21], which was first introduced in 1978 in the context of social network analysis [24]. Being parameterized relaxation of independent sets, co-t-plexes are also known as *$(t - 1)$-dependent* or *$(t - 1)$-stable sets* [15] and have plenty of applications in the field of computational biology [13], property testing [14, 22] and social network analysis [2, 4, 24].

A t-sparse set in a graph is *maximum* if the graph contains no larger t-sparse set. The *maximum t-sparse set problem* (MtSSP) asks for a maximum t-sparse set of a graph. In 1992, Djidjev *et al.* [11] studied this

Keywords. Outerplane graphs, treewidth, vector weights, t-sparse sets, maximum weight.

School of Mathematics, Southeast University, Nanjing 210096, P.R. China.
*Corresponding author: wslin@seu.edu.cn

© The authors. Published by EDP Sciences, ROADEF, SMAI 2023

This is an Open Access article distributed under the terms of the Creative Commons Attribution License (https://creativecommons.org/licenses/by/4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

problem under the name "maximum t-dependent set problem", where they proved that the decision version of this problem is NP-complete for general graphs and presented a linear-time algorithm for trees. Moreover, they also observed that this problem is solvable in linear time for graphs with constant treewidth if their tree-decomposition is given. The study was furthered in [10], in which the NP-completeness was demonstrated for bipartite planar graphs while the polynomial-time algorithms were designed for cographs, trees, split graphs and graphs with bounded treewidth.

In 2009, Havet *et al.* [17] proved that the MtSSP is NP-complete for unit-disk graphs and they provided a polynomial-time approximation scheme (PTAS) for unit-disk graphs. And then in 2010, constant factor approximation algorithms were also developed for unit-disk graphs in [1]. In 2014, Kumar *et al.* [18] presented a $(1+\frac{t}{t+1})$-approximation algorithm for the MtSSP on bipartite graphs. And an improvement to $1+\frac{t}{t+2}$ was provided by Hosseinian and Butenko [16] in 2022, which is the current best-known approximation ratio on bipartite graphs.

The 1-sparse set was also widely studied under the name "dissociation set" being introduced by Yannakakis [28] in 1981. The maximum 1-sparse set problem can help find a lower bound for the 1-improper chromatic number of a graph [17]. Moreover, its dual problem is known as the *minimum 3-path vertex cover problem*, the readers may refer to [25–27] for more details on 1-sparse sets. Furthermore, finding a maximum 2-sparse set of a graph also has several applications, for instance, in information dissemination in hypercubes with a large number of faulty processors [7].

The *maximum weight t-sparse set problem* (MWtSSP) is to find a t-sparse set of maximum weight in a weighted graph (G,w), where the weight function w is a mapping from $V(G)$ to $\mathbb{R}$ and the weight of a vertex set is defined to be the sum of the weights of its elements. As the MtSSP is a special case of MWtSSP where the weight of any vertex is 1, all the hardness results for MtSSP also hold for MWtSSP. In 2010, a greedy randomized adaptive search procedure (GRASP) algorithm was provided to find the maximum weight t-sparse set in [5], which can be used to find large profitable portfolios of stocks with high diversity.

The MWtSSP for $t=0$ is referred to as the *maximum weight independent set problem* (MWISP), which has many applications in several fields of computer science, such as scheduling, wireless networks, computer graphics, map labeling, molecular biology [3]. It is known that polynomial-time algorithms exist for the MWISP on trees [8], line graphs (where it becomes equivalent to the problem of finding a maximum weight matching), claw-free graphs [20], fork-free graphs [19], apple-free graphs [6] and graphs with bounded treewidth [9]. However, it remains NP-complete for graphs in some special classes including $K_{1,4}$-free graphs [20].

Although the MWISP has been extensively studied on some special graph classes, there are few studies on the MWtSSP as far as we know. Thus one might be interested in considering the MWtSSP in special classes of graphs. When considering the MWtSSP on outerplane graphs, we find that further generalization can bring some convenience to the solution. That is the primary motivation for us to define vector weights (refer to Def. 2.1) on graphs and then propose the following maximum weight t-sparse set problem on vector-weighted graphs in this paper.

Maximum Weight t-Sparse Set Problem on Vector-Weighted Graphs
Given a nonnegative integer t and a vector-weighted graph $(G,\vec{w},t)$,
Find A maximum weight t-sparse set of $(G,\vec{w},t)$.

The above problem can be regarded as the vector version of the MWtSSP. In this paper, we devote to solving this problem for outerplane graphs and graphs with bounded treewidth. The remainder of this paper is organized as follows. In Section 2, we introduce some relevant definitions and notation, including the vector weights. Then we present a dynamic programming algorithm in Section 3 to find a maximum weight t-sparse set on a 2-connected vector-weighted outerplane graph. Based on the algorithm in Section 3, Section 4 designs an algorithm that runs in $O((t+2)^4 n)$ time for a connected but not 2-connected vector-weighted outerplane graph $(G,\vec{w},t)$, where $n=|V(G)|$. Section 5 gives a dynamic programming algorithm that takes $O((k+1)(t+2)^{2k+2}n)$ time

to deal with the MWtSSP on vector-weighted graphs with treewidth at most k. Finally, Section 6 summarizes our works and suggests some future research directions.

2. Preliminaries

Let's start with designing the vector weights on graphs.

Definition 2.1. Given a nonnegative integer t, the vector-weighted graph $(G, \vec{w}, t)$ is a graph G with a vector weight function $\vec{w} : V(G) \to \mathbb{R}^{t+2}$, where $\vec{w}(v) = (w(v;-1), w(v;0), \ldots, w(v;t))$ for each $v \in V(G)$. Notice that the entries in vector $\vec{w}(v)$ are indexed by integers from -1 to t.

For convenience, we use the abbreviation $(G, \vec{w})$ for $(G, \vec{w}, t)$. For $S \subseteq V(G)$, let $d_S(G; v) = |N_G(v) \cap S|$ for $v \in S$ and $d_S(G; v) = -1$ for $v \in V(G) \setminus S$. The weight of a t-sparse set S in $(G, \vec{w})$ is defined as $\vec{w}(S, G) = \sum_{v \in V(G)} w(v; d_S(G; v))$. Here we note that the weight that a vertex v can provide to a t-sparse set S depends on the value of $d_S(G; v)$. In other words, the weight of a t-sparse set is closely related to the level of adjacency among its elements. A *maximum weight t-sparse set* in $(G, \vec{w})$ is a t-sparse set such that its weight is maximum among all t-sparse sets of $(G, \vec{w})$ and we use $\alpha^{(t)}_{\vec{w}}(G)$ to denote its weight.

From the above definitions, it is not difficult to see that the MWtSSP on the vector-weighted graph $(G, \vec{w})$ will be equivalent to the MWtSSP on the ordinary weighted graph (G, w) if $\vec{w}(v) = (0, w(v), \ldots, w(v))_{1\times(t+2)}$ for each $v \in V(G)$.

Given a nonnegative integer t and a vector-weighted graph $(G, \vec{w})$. Let $[t]^- = \{-1, 0, 1, \ldots, t\}$, $[t]^0 = \{0, 1, \ldots, t\}$, $[t]^1 = \{1, 2, \ldots, t\}$ and M be a very large positive number. For any $a \in V(G)$ and any $p \in [t]^-$, let $\mathbb{S}_{(G;a;p)} = \{S : S \subseteq V(G), \Delta(G[S]) \le t \text{ and } d_S(G; a) = p\}$ and

$$\alpha^{(t)}_{\vec{w}}(G; a; p) = \begin{cases} \max\{\vec{w}(S, G) : S \in \mathbb{S}_{(G;a;p)}\}, & \text{if } \mathbb{S}_{(G;a;p)} \neq \emptyset, \\ -M, & \text{otherwise.} \end{cases}$$

For any two distinct vertices $a, b \in V(G)$ and any two integers $p, q \in [t]^-$, let $\mathbb{S}_{(G;a,b;p,q)} = \mathbb{S}_{(G;a;p)} \cap \mathbb{S}_{(G;b;q)}$ and

$$\alpha^{(t)}_{\vec{w}}(G; a, b; p, q) = \begin{cases} \max\{\vec{w}(S, G) : S \in \mathbb{S}_{(G;a,b;p,q)}\}, & \text{if } \mathbb{S}_{(G;a,b;p,q)} \neq \emptyset, \\ -M, & \text{otherwise.} \end{cases}$$

And for any three distinct vertices $a, b, c \in V(G)$ and any three integers $p, q, r \in [t]^-$, let $\mathbb{S}_{(G;a,b,c;p,q,r)} = \mathbb{S}_{(G;a;p)} \cap \mathbb{S}_{(G;b;q)} \cap \mathbb{S}_{(G;c;r)}$ and

$$\alpha^{(t)}_{\vec{w}}(G; a, b, c; p, q, r) = \begin{cases} \max\{\vec{w}(S, G) : S \in \mathbb{S}_{(G;a,b,c;p,q,r)}\}, & \text{if } \mathbb{S}_{(G;a,b,c;p,q,r)} \neq \emptyset, \\ -M, & \text{otherwise.} \end{cases}$$

It can be easily seen that for any three distinct vertices $a, b, c \in V(G)$, there are

$$\begin{aligned} \alpha^{(t)}_{\vec{w}}(G) &= \max_{p\in[t]^-} \left\{ \alpha^{(t)}_{\vec{w}}(G; a; p) \right\} \\ &= \max_{p\in[t]^-} \left\{ \max_{q\in[t]^-} \left\{ \alpha^{(t)}_{\vec{w}}(G; a, b; p, q) \right\} \right\} \\ &= \max_{p\in[t]^-} \left\{ \max_{q\in[t]^-} \left\{ \max_{r\in[t]^-} \left\{ \alpha^{(t)}_{\vec{w}}(G; a, b, c; p, q, r) \right\} \right\} \right\}. \end{aligned} \tag{1}$$

A graph that can be embedded in the plane is called a *planar graph*, and a fixed planar embedding of a planar graph is called a *plane graph*. And an *outerplane graph* is a plane graph in which all vertices are incident with *the outer face*. For an outerplane graph G, we use f_0 to denote its outer face and $\partial(f)$ to denote the boundary of a face f in G.

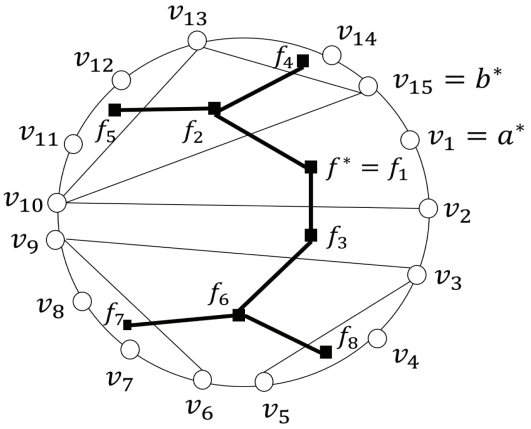

FIGURE 1. An example of G and $T^*_{f_1}$.

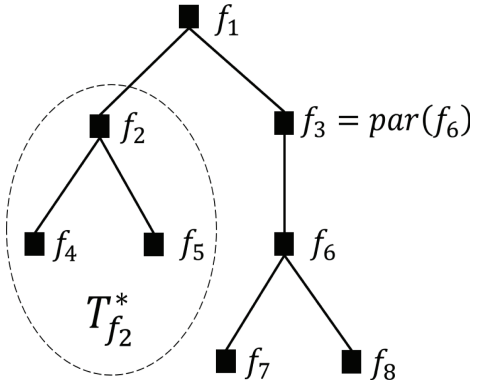

FIGURE 2. $T^*_{f_1}$ and its subtree.

3. The MWtSSP for 2-connected vector-weighted outerplane graphs

This section deals with the MWtSSP for 2-connected vector-weighted outerplane graphs, and the general case will be addressed in the next section.

Let G be a 2-connected outerplane graph. Removing from the dual of G the vertex corresponding to f_0 results in a tree, which is denoted by T^*. Let a^*b^* be an edge in $\partial(f_0)$ and f_1 be an interior face of G with $a^*b^* \in \partial(f_1)$. By performing a *breadth-first search* (BFS) on T^* starting from f_1, we obtain an order of the vertices of T^* according to the time each vertex being visited. Suppose T^* has z interior faces. We denote this order by $f_1, f_2, \ldots, f_z$. To specify f_1 as the root of T^*, we use $T^*_{f_1}$ to denote the tree T^* rooted at f_1. For convenience, we shall default that $f_1, f_2, \ldots, f_z$ represent both vertices of $T^*_{f_1}$ and the interior faces of G (please refer to Figs. 1 and 2).

Notice that any two adjacent interior faces f_i and f_j in an outerplane graph share exactly one edge, we denote this edge by $e(f_i, f_j)$. For any $f_i \in V(T^*_{f_1})$, let $par(f_i)$ be the parent of f_i in $T^*_{f_1}$ and $T^*_{f_i}$ be the subtree of $T^*_{f_1}$ consisting of f_i and all its descendants (see Fig. 2, $par(f_6) = f_3$ and $T^*_{f_2} = T^*[\{f_2, f_4, f_5\}]$). Moreover, let V_{f_i} be the collection of all vertices on the faces of G that corresponding to vertices in $T^*_{f_i}$ (see Fig. 1, $V_{f_2} = \{v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}\}$). And let H_i denote the induced subgraph $G[V_{f_i}]$ (see Fig. 3, $H_2 = G[V_{f_2}]$). Clearly, $G = H_1$.

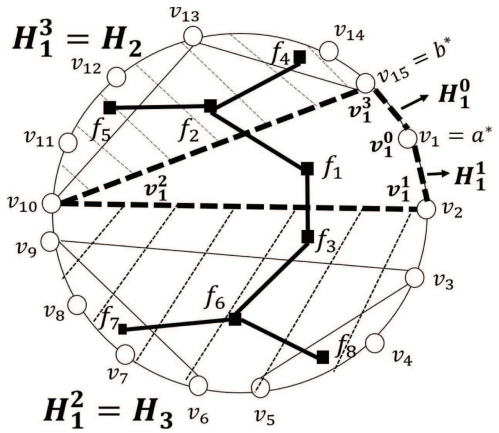

FIGURE 3. The case of $f_i = f_1$.

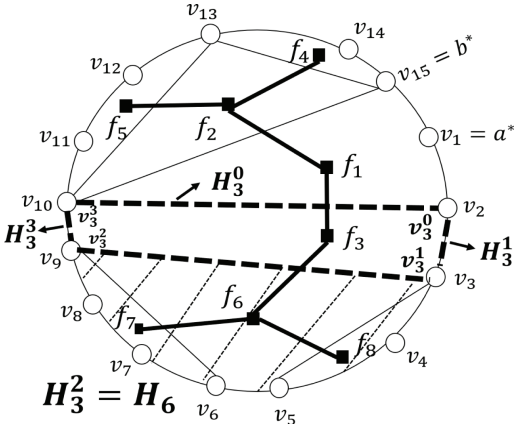

FIGURE 4. The case of $f_i \neq f_1$.

For each $i \in [z]^1$, let $h_i = d_G(f_i) - 1$ and let the vertices on the boundary of f_i be $v_i^0, v_i^1, \ldots, v_i^{h_i}$ in clockwise order, where $v_i^0 v_i^{h_i} = a^* b^*$ if $f_i = f_1$ (see Fig. 3, $v_1^0 v_1^3 = a^* b^*$) and $v_i^0 v_i^{h_i} = e(f_i, par(f_i))$ if $f_i \neq f_1$ (see Fig. 4, $v_3^0 v_3^3 = e(f_3, f_1) = v_2 v_{10}$). And let $H_i^0 = v_i^0 v_i^{h_i}$ for each $i \in [z]^1$. For each $j \in [h_i]^1$, if there exists $f_k \in V(T_{f_1}^*)$ such that $e(f_i, f_k) = v_i^{j-1} v_i^j$, then let $H_i^j = H_k$; otherwise, let $H_i^j = v_i^{j-1} v_i^j$. Then it is not difficult to see that $H_i = \cup_{j=0}^{h_i} H_i^j = G[V_{f_i}]$. (see Fig. 3, $H_1^2 = H_3$ since $e(f_1, f_3) = v_2 v_{10}$, and $H_1^1 = v_1^0 v_1^1 = v_1 v_2$.)

Let us proceed to give two definitions that are also useful in the description of our algorithms. For any two integers $p, q \in [t]^-$, let

$$\varepsilon(p,q) = \begin{cases} (p+1)(q+1)M, & p,q \in \{-1,0\}, \\ (p-1)M + (q-1)M, & p,q \in [t]^1, \\ M, & \text{otherwise;} \end{cases}$$

$$\eta(p,q) = \begin{cases} (p+1)(q+1)M, & p,q \in \{-1,0\}, \\ (p+1)M, & p \in \{-1,0\} \text{ and } q \in [t]^1, \\ (q+1)M, & p \in [t]^1 \text{ and } q \in \{-1,0\}, \\ 0, & p,q \in [t]^1. \end{cases}$$

With these definitions at hand, we have the following lemma.

Lemma 3.1. *Given a nonnegative integer t and a 2-connected vector-weighted outerplane graph $(G,\vec{w})$. Let f_i be an interior face of G. Then $\{\alpha^{(t)}_{\vec{w}}(H_i;v_i^0,v_i^{h_i};p,q):p,q\in[t]^-\}$ can be calculated in $O(d_G(f_i)(t+2)^4)$ time if $\{\alpha^{(t)}_{\vec{w}}(H_i^j;v_i^{j-1},v_i^j;p,q):p,q\in[t]^-\}$ is known for each $j\in[h_i]^1$.*

Proof. Let $\tilde{H}_i^j=\cup_{x=1}^j H_i^x$ for each $j\in[h_i]^1$. Then $\tilde{H}_i^j=\tilde{H}_i^{j-1}\cup H_i^j$ and $V(\tilde{H}_i^{j-1})\cap V(H_i^j)=\{v_i^{j-1}\}$ for $j\in\{2,3,\ldots,h_i\}$. Moreover, for each $j\in\{2,3,\ldots,h_i\}$ and any $p,q\in[t]^-$, we have

$$\alpha^{(t)}_{\vec{w}}\Big(\tilde{H}_i^j;v_i^0,v_i^j;p,q\Big)=\max_{r\in[t]^-}\Big\{\alpha^{(t)}_{\vec{w}}\Big(\tilde{H}_i^j;v_i^0,v_i^j,v_i^{j-1};p,q,r\Big)\Big\},$$

where

$$\alpha^{(t)}_{\vec{w}}\Big(\tilde{H}_i^j;v_i^0,v_i^j,v_i^{j-1};p,q,-1\Big)=\alpha^{(t)}_{\vec{w}}\Big(\tilde{H}_i^{j-1};v_i^0,v_i^{j-1};p,-1\Big)+\alpha^{(t)}_{\vec{w}}\Big(H_i^j;v_i^{j-1},v_i^j;-1,q\Big)-w\Big(v_i^{j-1};-1\Big),$$

and for each $r\in[t]^0$,

$$\begin{aligned}\alpha^{(t)}_{\vec{w}}\Big(\tilde{H}_i^j;v_i^0,v_i^j,v_i^{j-1};p,q,r\Big)=\max_{s\in[r]^0}\Big\{&\alpha^{(t)}_{\vec{w}}\Big(\tilde{H}_i^{j-1};v_i^0,v_i^{j-1};p,s\Big)+\alpha^{(t)}_{\vec{w}}\Big(H_i^j;v_i^{j-1},v_i^j;r-s,q\Big)\\&-w\Big(v_i^{j-1};s\Big)-w\Big(v_i^{j-1};r-s\Big)+w\Big(v_i^{j-1};r\Big)\Big\}.\end{aligned}$$

As the calculation of $\{\alpha^{(t)}_{\vec{w}}(\tilde{H}_i^j;v_i^0,v_i^j;p,q):p,q\in[t]^-\}$ takes $O((t+2)^4)$ time for each $j\in\{2,3,\ldots,h_i\}$, $\{\alpha^{(t)}_{\vec{w}}(\tilde{H}_i^{h_i};v_i^0,v_i^{h_i};p,q):p,q\in[t]^-\}$ can be computed in $O((h_i-1)(t+2)^4)$ time. And then, notice that $H_i=H_i^0\cup\tilde{H}_i^{h_i}$, $H_i^0=v_i^0v_i^{h_i}$ and $E(H_i^0)\cap E(\tilde{H}_i^{h_i})=\emptyset$. We immediately have that, for any $p,q\in[t]^-$,

$$\alpha^{(t)}_{\vec{w}}\Big(H_i;v_i^0,v_i^{h_i};p,q\Big)=\begin{cases}\alpha^{(t)}_{\vec{w}}\Big(\tilde{H}_i^{h_i};v_i^0,v_i^{h_i};p-1,q-1\Big)-w\big(v_i^0;p-1\big)\\ -w\Big(v_i^{h_i};q-1\Big)+w\big(v_i^0;p\big)+w\Big(v_i^{h_i};q\Big), & \text{if } p,q\in[t]^1,\\ \alpha^{(t)}_{\vec{w}}\Big(\tilde{H}_i^{h_i};v_i^0,v_i^{h_i};p,q\Big)-\eta(p,q), & \text{otherwise.}\end{cases}$$

This takes $O((t+2)^2)$ time. So the total time is $O(h_i(t+2)^4)=O(d_G(f_i)(t+2)^4)$. □

The idea to solve the MWtSSP on a 2-connected outerplane graph $(G,\vec{w})$ is as follows. Let $f_1,f_2,\ldots,f_z$ be the interior faces of G and a^*b^* be an edge in $\partial(f_0)\cap\partial(f_1)$. Firstly, we assign initial values to all edges lying on the boundary of f_0 except a^*b^*. Secondly, we compute $\{\alpha^{(t)}_{\vec{w}}(H_i;v_i^0,v_i^{h_i};p,q):p,q\in[t]^-\}$ for i from z to 1 according to Lemma 3.1. Lastly, we output $\{\alpha^{(t)}_{\vec{w}}(G;a^*;p):p\in[t]^-\}$ from $\{\alpha^{(t)}_{\vec{w}}(H_1;v_1^0,v_1^{h_1};p,q):p,q\in[t]^-\}$ by equation (1). The details are in the following algorithm.

Theorem 3.2. *For any nonnegative integer t, the maximum weight t-sparse set problem for 2-connected vector-weighted outerplane graphs can be solved in $O((t+2)^4n)$ time.*

Proof. The correctness of Algorithm 1 follows directly from Lemma 3.1. Next, we analyze its time complexity.

Firstly, there are n edges on the boundary of f_0, so it takes $O((n-1)(t+2)^2)$ time to assign initial values to the edges in $\partial(f_0)$ except a^*b^*. Secondly, Lemma 3.1 shows that one can compute $\{\alpha^{(t)}_{\vec{w}}(H_i;v_i^0,v_i^{h_i};p,q):p,q\in[t]^-\}$ in $O(d_G(f_i)(t+2)^4)$ time for each $i\in[z]^1$. Thus, within $O(\sum_{f_i\in V(T^*_{f_1})}d_G(f_i)(t+2)^4)=O(m(t+2)^4)$ time, we can get $\{\alpha^{(t)}_{\vec{w}}(H_1;a^*,b^*;p,q):p,q\in[t]^-\}$. Thirdly, $\{\alpha^{(t)}_{\vec{w}}(G;a^*;p):p\in[t]^-\}$ can be obtained easily in $O(t+2)$ time. Note that every simple 2-connected outerplane graph other than K_2 has a vertex of degree two, one can deduce that $m\le 2n$. Hence the time complexity of Algorithm 1 is $O((t+2)^4n)$.

Finally, equation (1) implies that the calculation of $\alpha^{(t)}_{\vec{w}}(G)$ takes $O(t+2)$ time according to the output results of Algorithm 1. Therefore, the MWtSSP for 2-connected vector-weighted outerplane graphs can be solved in $O((t+2)^4n)$ time. The theorem is proved. □

Algorithm 1. The MWtSSP for 2-connected vector-weighted outerplane graphs.

Input: a nonnegative integer t, a 2-connected vector-weighted outerplane graph $(G, \vec{w})$ with the interior faces $f_1, f_2, \ldots, f_z$ and $a^*b^* \in \partial(f_0) \cap \partial(f_1)$;

Output: $\{\alpha_{\vec{w}}^{(t)}(G; a^*; p) : p \in [t]^-\}$;

for each $ab \in \partial(f_0) \setminus \{a^*b^*\}$ **do**
  for $p : -1 \to t$ **do**
    for $q : -1 \to t$ **do**
      Set $\alpha_{\vec{w}}^{(t)}(ab; a, b; p, q) := w(a; p) + w(b; q) - \varepsilon(p, q)$
    end for
  end for
end for

for $i : z \to 1$ **do**
  Compute $\{\alpha_{\vec{w}}^{(t)}(H_i; v_i^0, v_i^{h_i}; p, q) : p, q \in [t]^-\}$ according to Lemma 3.1
end for

for $p : -1 \to t$ **do**
  Set $\alpha_{\vec{w}}^{(t)}(G; a^*; p) := \max_{q \in [t]^-}\{\alpha_{\vec{w}}^{(t)}(H_1; a^*, b^*; p, q)\}$
end for

Output $\{\alpha_{\vec{w}}^{(t)}(G; a^*; p) : p \in [t]^-\}$

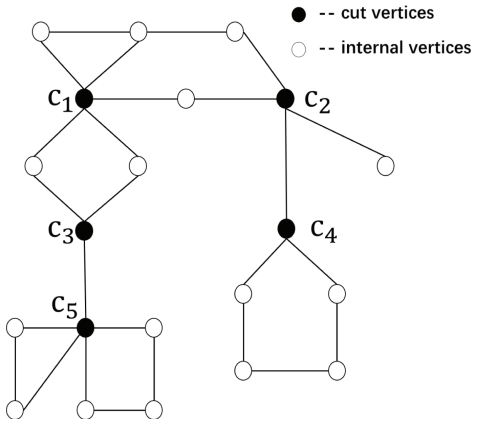

FIGURE 5. An outerplane graph G.

4. The MWtSSP for connected vector-weighted outerplane graphs

In this section, we solve the MWtSSP on outerplane graphs which are connected but not 2-connected. We begin with the following definitions.

Given a connected outerplane graph G. The *blocks* of G fit together in a treelike structure which is called the *block tree* of G and denoted by T_G. For convenience, we default that B (resp. c) represents both a block (resp. a *cut vertex*) of G and a vertex of T_G. A block B and a cut vertex c being adjacent in T_G if and only if B contains c in G. And the blocks of G which correspond to leaves of T_G are referred to as its *end blocks*. In particular, we use T_{B_r} to denote T_G rooted at a block B_r of G. For example, the blocks of the outerplane graph G in Figure 5 are indicated in Figure 6. And choose the block B_1 as root, the rooted block tree T_{B_1} is shown in Figure 7.

Let $2l = \max\{d_{T_{B_r}}(v, r) : v \in V(T_{B_r})\}$ and $L_i = \{v \in V(T_{B_r}) : d_{T_{B_r}}(v, r) = i\}$ for each $i \in [2l]^0$. Apparently, each $v \in \cup_{i \in [l]^0} L_{2i}$ corresponds to a block in G while each $v \in \cup_{i \in [l-1]^0} L_{2i+1}$ corresponds to a cut vertex in G. Notice that any block of G is either a 2-connected outerplane graph or a complete graph K_2. For any block

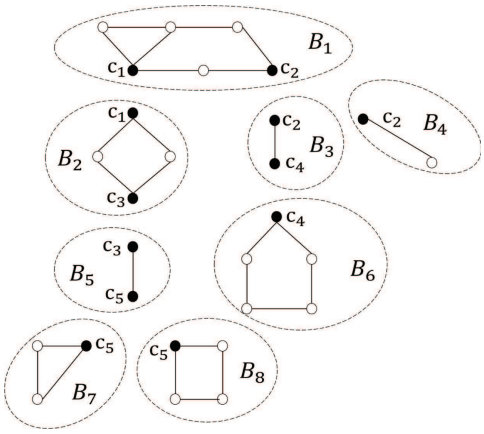

FIGURE 6. The blocks of the graph G.

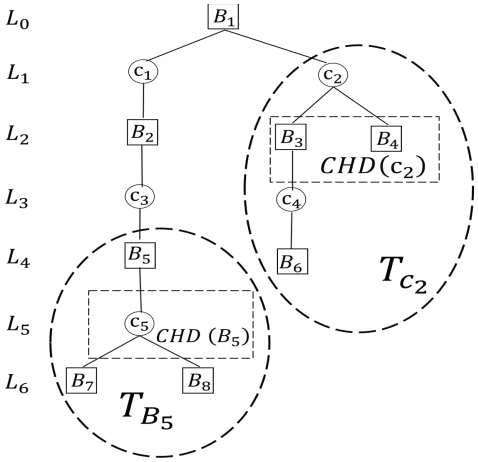

FIGURE 7. The block tree T_{B_1} and its subtrees.

B of G, we denote by $par(B)$ the parent of B in T_{B_r} and CHD(B) the set of children of B in T_{B_r}. Moreover, let T_B represent the subtree of T_{B_r} consisting of B and all its descendants (see Fig. 7) and R_B represent the subgraph of G induced by the vertices of all blocks in T_B (see Fig. 8). Similarly, for any cut vertex c of G, we denote by $par(c)$ the parent of c in T_{B_r} and CHD(c) the set of children of c in T_{B_r}. And let T_c represent the subtree of T_{B_r} consisting of c and all its descendants (see Fig. 7) and Q_c represent the subgraph of G induced by the vertices of all blocks in T_c (see Fig. 8).

For a complete graph $(K_2, \vec{w})$ with $V(K_2) = \{a, b\}$, the following algorithm will output $\{\alpha^{(t)}_{\vec{w}}(G; a; p) : p \in [t]^-\}$ in $O(t+2)$ time.

The above algorithm is correct. Meanwhile, one can deduce the following fact.

Claim 1. Let B be an end block of $(G, \vec{w})$. If $B = K_2$, then $\{\alpha^{(t)}_{\vec{w}}(B; par(B); p) : p \in [t]^-\}$ can be obtained in $O(t+2)$ time through Algorithm 2. And if $B \neq K_2$, then the calculations of $\{\alpha^{(t)}_{\vec{w}}(B; par(B); p) : p \in [t]^-\}$ take $O(|V(B)|(t+2)^4)$ time by Algorithm 1.

We now turn our attention to the blocks which are not end blocks.

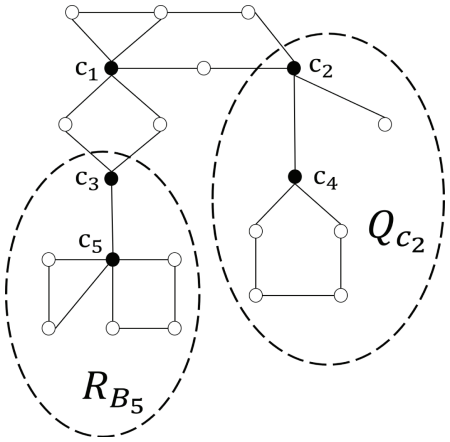

FIGURE 8. The induced subgraphs of G.

Algorithm 2. The MWtSSP for $(K_2, \vec{w})$.

Input: a nonnegative integer t and $(K_2, \vec{w})$ with $V(K_2) = \{a, b\}$;
Output: $\{\alpha^{(t)}_{\vec{w}}(K_2; a; p) : p \in [t]^-\}$;
1: Set $\alpha^{(t)}_{\vec{w}}(K_2; a; -1) := w(a; -1) + \max\{w(b; -1), w(b; 0)\}$
2: Set $\alpha^{(t)}_{\vec{w}}(K_2; a; 0) := w(a; 0) + w(b; -1)$
3: Set $\alpha^{(t)}_{\vec{w}}(K_2; a; 1) := w(a; 1) + w(b; 1)$
4: **for** $p : 2 \to t + 2$ **do**
5: Set $\alpha^{(t)}_{\vec{w}}(K_2; a; p) := -M$
6: **end for**
7: **Output** $\{\alpha^{(t)}_{\vec{w}}(K_2; a; p) : p \in [t]^-\}$

Lemma 4.1. *Let B be a block of $(G, \vec{w})$. Suppose that B is not an end block. If $\{\alpha^{(t)}_{\vec{w}}(Q_c; c; p) : p \in [t]^-\}$ is known for each $c \in \mathrm{CHD}(B)$, then $\{\alpha^{(t)}_{\vec{w}}(R_B; par(B); p) : p \in [t]^-\}$ can be calculated in $O(|V(B)|(t + 2)^4)$ time.*

Proof. Please refer to Figure 9 for the structure of $(R_B, \vec{w})$.

For each $c \in \mathrm{CHD}(B)$, let $w^*(c; -1) = \alpha^{(t)}_{\vec{w}}(Q_c; c; -1)$ and for each $p \in [t]^0$, let

$$w^*(c; p) = \max_{q \in [t-p]^0} \Big\{\alpha^{(t)}_{\vec{w}}(Q_c; c; q) - w(c; q) + w(c; p + q)\Big\}.$$

And for any $v \in V(B)$, we define $\vec{w}^*(v)$ as follows:

$$\vec{w}^*(v) = \begin{cases} (w^*(c; -1), w^*(v; 0), \ldots, w^*(v; t)), & v \in \mathrm{CHD}(B), \\ \vec{w}(v), & v \in V(B) \setminus \mathrm{CHD}(B). \end{cases} \tag{2}$$

Then $(B, \vec{w}^*)$ is also a 2-connected vector-weighted outerplane graph. Let us first demonstrate the following claim.

Claim 2. For each $p \in [t]^-$, we have

$$\alpha^{(t)}_{\vec{w}^*}(B; par(B); p) = \alpha^{(t)}_{\vec{w}}(R_B; par(B); p). \tag{3}$$

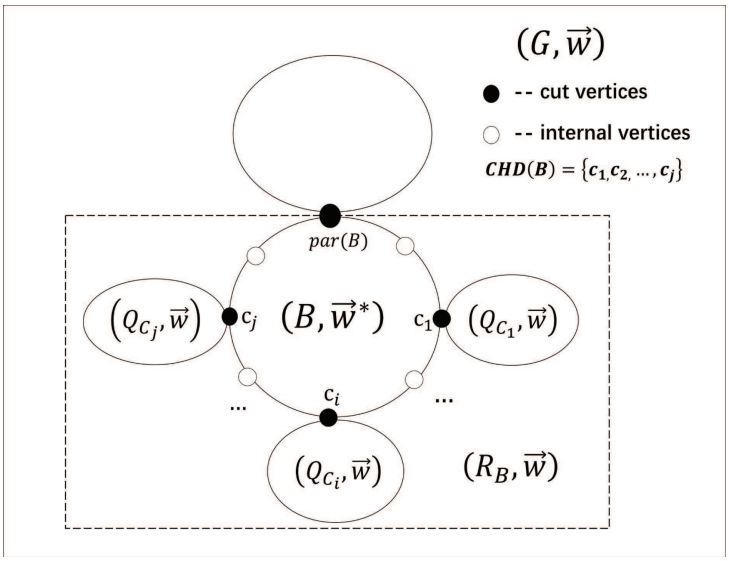

FIGURE 9. The structure of $(R_B, \vec{w})$.

Proof. Given a fixed integer $p \in [t]^-$, let T_p^* be a maximum weight t-sparse set of $(B, \vec{w}^*)$ with $d_{T_p^*}(B; par(B)) = p$. Then we have $\vec{w}^*(T_p^*, B) = \alpha_{\vec{w}^*}^{(t)}(B; par(B); p)$. And for each $v \in V(B)$, let $t_p^*(v) = d_{T_p^*}(B; v)$.

Now we construct a t-sparse set T_p of $(R_B, \vec{w})$ satisfying the following three conditions:

(1) $T_p \cap V(B) = T_p^*$;
(For each $v \in V(R_B)$, let $t_p(v) = d_{T_p}(R_B; v)$.)
(2) $\vec{w}(T_p \cap V(Q_c), Q_c) = w^*(c; -1)$ for each $c \in \mathrm{CHD}(B) \setminus T_p^*$;
(3) $\vec{w}(T_p \cap V(Q_c), Q_c) - w(c; t_p(c) - t_p^*(c)) + w(c; t_p(c)) = w^*(c; t_p^*(c))$ for each $c \in \mathrm{CHD}(B) \cap T_p^*$.

Due to the definition of $(B, \vec{w}^*)$ we can always find the desired t-sparse set T_p in $(R_B, \vec{w})$. We next prove that $\vec{w}(T_p, R_B) = \alpha_{\vec{w}^*}^{(t)}(B; par(B); p)$.

$$
\begin{aligned}
\vec{w}(T_p, R_B) &= \sum_{v \in V(R_B)} w(v; t_p(v)) = \sum_{v \in V(B) \setminus \mathrm{CHD}(B)} w(v; t_p(v)) + \sum_{c \in \mathrm{CHD}(B)} \left[\sum_{v \in V(Q_c)} w(v; t_p(v)) \right] \\
&= \sum_{v \in V(B) \setminus \mathrm{CHD}(B)} w^*(v; t_p(v)) + \sum_{\substack{c \in \mathrm{CHD}(B) \\ t_p(c) = -1}} \left[\sum_{v \in V(Q_c)} w(v; t_p(v)) \right] + \sum_{\substack{c \in \mathrm{CHD}(B) \\ t_p(c) \in [t]^0}} \left[\sum_{v \in V(Q_c)} w(v; t_p(v)) \right] \\
&= \sum_{v \in V(B) \setminus \mathrm{CHD}(B)} w^*\big(v; t_p^*(v)\big) + \sum_{\substack{c \in \mathrm{CHD}(B) \\ t_p^*(c) = -1}} \vec{w}(T_p \cap V(Q_c), Q_c) \\
&\quad + \sum_{\substack{c \in \mathrm{CHD}(B) \\ t_p^*(c) \in [t]^0}} \left[\vec{w}(T_p \cap V(Q_c), Q_c) - w\big(c; t_p(c) - t_p^*(c)\big) + w(c; t_p(c)) \right] \\
&= \sum_{v \in V(B) \setminus \mathrm{CHD}(B)} w^*\big(v; t_p^*(v)\big) + \sum_{\substack{c \in \mathrm{CHD}(B) \\ t_p^*(c) = -1}} w^*(c; -1) + \sum_{\substack{c \in \mathrm{CHD}(B) \\ t_p^*(c) \in [t]^0}} w^*\big(c; t_p^*(c)\big)
\end{aligned}
$$

$$= \sum_{v\in V(B)} w^*\big(v; t_p^*(v)\big) = \vec{w}^*\big(T_p^*, B\big) = \alpha_{\vec{w}^*}^{(t)}(B; par(B); p).$$

Next, we show that $\vec{w}(T_p, R_B) = \alpha_{\vec{w}}^{(t)}(R_B; par(B); p)$. Without loss of generality, let S_p be a t-sparse set of $(R_B, \vec{w})$ with $d_{S_p}(R_B; par(B)) = p$. We shall prove that $\vec{w}(S_p, R_B) \le \vec{w}(T_p, R_B)$.

Let $S_p^* = S_p \cap V(B)$. Clearly, S_p^* is a t-sparse set of $(B, \vec{w}^*)$ with $d_{S_p^*}(B; par(B)) = p$, which implies $\vec{w}^*(S_p^*, B) \le \vec{w}^*(T_p^*, B)$ since T_p^* is a maximum weight t-sparse set of $(B, \vec{w}^*)$ with $d_{T_p^*}(B; par(B)) = p$. And let $s_p(v) = d_{S_p}(R_B; v)$ for each $v \in V(R_B)$ and $s_p^*(v) = d_{S_p^*}(B; v)$ for each $v \in V(B)$. We immediately have

$$\begin{aligned}
\vec{w}(S_p, R_B) &= \sum_{v\in V(B)\setminus \mathrm{CHD}(B)} w^*\big(v; s_p^*(v)\big) + \sum_{\substack{c\in \mathrm{CHD}(B)\\ s_p^*(c)=-1}} \vec{w}(S_p \cap V(Q_c), Q_c) \\
&\quad + \sum_{\substack{c\in \mathrm{CHD}(B)\\ s_p^*(c)\in[t]^0}} \big[\vec{w}(S_p \cap V(Q_c), Q_c) - w\big(c; s_p(c) - s_p^*(c)\big) + w(c; s_p(c))\big] \\
&\le \sum_{v\in V(B)\setminus \mathrm{CHD}(B)} w^*\big(v; s_p^*(v)\big) + \sum_{\substack{c\in \mathrm{CHD}(B)\\ s_p^*(c)=-1}} \alpha_{\vec{w}}^{(t)}(Q_c; c; -1) \\
&\quad + \sum_{\substack{c\in \mathrm{CHD}(B)\\ s_p^*(c)\in[t]^0}} \left[\max_{q\in\left[t-s_p^*(c)\right]^0} \left\{\alpha_{\vec{w}}^{(t)}(Q_c; c; q) - w(c; q) + w\big(c; s_p^*(c)+q\big)\right\}\right] \\
&= \sum_{v\in V(B)\setminus \mathrm{CHD}(B)} w^*\big(v; s_p^*(v)\big) + \sum_{\substack{c\in \mathrm{CHD}(B)\\ s_p^*(c)=-1}} w^*(c; -1) + \sum_{\substack{c\in \mathrm{CHD}(B)\\ s_p^*(c)\in[t]^0}} w^*\big(c; s_p^*(c)\big) \\
&= \vec{w}^*\big(S_p^*, B\big) \le \vec{w}^*\big(T_p^*, B\big) = \vec{w}(T_p, R_B).
\end{aligned}$$

Hence it can be verified easily, by the arbitrariness of S_p, that T_p is a maximum weight t-sparse set of $(R_B, \vec{w})$ with $d_{T_p}(R_B; par(B)) = p$. Thus we have $\vec{w}(T_p, R_B) = \alpha_{\vec{w}}^{(t)}(R_B; par(B); p)$. Therefore, $\alpha_{\vec{w}^*}^{(t)}(B; par(B); p) = \alpha_{\vec{w}}^{(t)}(R_B; par(B); p)$ for any $p \in [t]^-$. The claim holds. □

The above claim reduces the problem of calculating $\{\alpha_{\vec{w}}^{(t)}(R_B; par(B); p) : p \in [t]^-\}$ on $(R_B, \vec{w})$ to that of computing $\{\alpha_{\vec{w}^*}^{(t)}(B; par(B); p) : p \in [t]^-\}$ on $(B, \vec{w}^*)$. This is desirable because when B is 2-connected, the latter problem can be solved by Algorithm 1, and when B is isomorphic to K_2, it can be solved by Algorithm 2. Thus, one can compute $\{\alpha_{\vec{w}}^{(t)}(R_B; par(B); p) : p \in [t]^-\}$ within $O(|V(B)|(t+2)^4)$ time when $\{\alpha_{\vec{w}}^{(t)}(Q_c; c; p) : p \in [t]^-\}$ is provided for each $c \in \mathrm{CHD}(B)$. This concludes the proof. □

In what is to follow, we will show how to deal with $(Q_c, \vec{w})$ for any cut vertex $c \in V(G)$.

Lemma 4.2. *Let c be a cut vertex of $(G, \vec{w})$. If $\{\alpha_{\vec{w}}^{(t)}(R_B; c; p) : p \in [t]^-\}$ is known for each $B \in \mathrm{CHD}(c)$, then $\{\alpha_{\vec{w}}^{(t)}(Q_c; c; p) : p \in [t]^-\}$ can be calculated in $O(d_G(c)(t+1)^2)$ time.*

Proof. Suppose that $\mathrm{CHD}(c) = \{B_1, B_2, \ldots, B_j\}$. Observe that c belongs to a t-sparse set S of $(Q_c, \vec{w})$ if and only if it belongs to $S \cap V(R_{B_i})$ for each $B_i \in \mathrm{CHD}(c)$, which implies that

$$\alpha_{\vec{w}}^{(t)}(Q_c; c; -1) = \sum_{i=1}^{j} \Big[\alpha_{\vec{w}}^{(t)}(R_{B_i}; c; -1) - w(c; -1)\Big] + w(c; -1),$$

$$\alpha_{\vec{w}}^{(t)}(Q_c; c; 0) = \sum_{i=1}^{j} \Big[\alpha_{\vec{w}}^{(t)}(R_{B_i}; c; 0) - w(c; 0)\Big] + w(c; 0).$$

For each $p \in [t]^0$ and each $i \in [j]^1$, let

$$\Delta(i;p) = \Big[\alpha^{(t)}_{\vec{w}}(R_{B_i};c;p) - w(c;p)\Big] - \Big[\alpha^{(t)}_{\vec{w}}(R_{B_i};c;0) - w(c;0)\Big]. \tag{4}$$

It is easy to see that $\Delta(i;0) = 0$ for any $i \in [j]^1$.

For any $j+1$ integers $x_1, x_2, \ldots, x_j, p$ in $[t]^0$ with $x_1+x_2+\cdots+x_j = p$, let $w(x_1, x_2, \ldots, x_j; p) = \max\{\vec{w}(S, Q_c) : S \subseteq V(Q_c), \Delta(Q_c[S]) \leq t, d_S(Q_c;c) = p \text{ and } d_S(R_{B_i};c) = x_i \text{ for each } i \in [j]^1\}$. Then for any fixed integer $p \in [t]^0$, we have

$$\alpha^{(t)}_{\vec{w}}(Q_c;c;p) = \max\big\{w(x_1, x_2, \ldots, x_j; p) : x_i \in [t]^0 \text{ for } i \in [j]^1 \text{ and } x_1 + x_2 + \cdots + x_j = p\big\}. \tag{5}$$

Moreover, for each $p \in [t]^0$ and each $q \in [j]^1$, we define

$$f(q;p) = \max\big\{w(x_1, x_2, \ldots, x_j; p) : x_i \in [t]^0 \text{ for } i \in [j]^1 \text{ and } x_1 + x_2 + \cdots + x_q = p\big\}. \tag{6}$$

Then by equations (5) and (6), it is not difficult to see that

$$\alpha^{(t)}_{\vec{w}}(Q_c;c;p) = f(j;p), \quad \text{for any } p \in [t]^0. \tag{7}$$

From the definition of $w(x_1, x_2, \ldots, x_j; p)$, its value can be computed as follows.

$$\begin{aligned}
w(x_1, x_2, \ldots, x_j; p) &= \sum_{i=1}^{j}\Big[\alpha^{(t)}_{\vec{w}}(R_{B_i};c;x_i) - w(c;x_i)\Big] + w(c;p) \\
&= \sum_{i=1}^{j}\Big[\Delta(i;x_i) + \alpha^{(t)}_{\vec{w}}(R_{B_i};c;0) - w(c;0)\Big] + w(c;p) \\
&= \sum_{i=1}^{j}\Delta(i;x_i) + \sum_{i=1}^{j}\Big[\alpha^{(t)}_{\vec{w}}(R_{B_i};c;0) - w(c;0)\Big] + w(c;p) \\
&= \sum_{i=1}^{j}\Delta(i;x_i) + \alpha^{(t)}_{\vec{w}}(Q_c;c;0) - w(c;0) + w(c;p).
\end{aligned}$$

Thus for any $p \in [t]^0$, by equation (6) and $w(x_1, x_2, \ldots, x_j; p)$, we have

$$f(1;p) = \Delta(1;p) + \alpha^{(t)}_{\vec{w}}(Q_c;c;0) - w(c;0) + w(c;p). \tag{8}$$

And for any $q \in \{2, 3, \ldots, j\}$,

$$\begin{aligned}
f(q;p) &= \max_{x_1+\cdots+x_q=p}\left\{\sum_{i=1}^{q}\Delta(i;x_i) + \sum_{i=q+1}^{j}\Delta(i;0) + \alpha^{(t)}_{\vec{w}}(Q_c;c;0) - w(c;0) + w(c;p)\right\} \\
&= \max_{x_1+\cdots+x_q=p}\left\{\sum_{i=1}^{q-1}\Delta(i;x_i) + \Delta(q;x_q) + \alpha^{(t)}_{\vec{w}}(Q_c;c;0) - w(c;0) + w(c;p)\right\} \\
&= \max_{x_1+\cdots+x_q=p}\left\{\sum_{i=1}^{q-1}\Delta(i;x_i) + \alpha^{(t)}_{\vec{w}}(Q_c;c;0) - w(c;0) + w(c;p-x_q)\right. \\
&\qquad \left. + \Delta(q;x_q) - w(c;p-x_q) + w(c;p)\right\}
\end{aligned}$$

$$= \max_{x_q \in [p]^0} \left\{ \max_{x_1+\cdots+x_{q-1}=p-x_q} \left\{ \sum_{i=1}^{q-1} \Delta(i;x_i) + \alpha_{\vec{w}}^{(t)}(Q_c;c;0) - w(c;0) + w(c;p-x_q) \right\} \right.$$
$$\left. + \Delta(q;x_q) - w(c;p-x_q) + w(c;p) \right\}$$
$$= \max_{x_q \in [p]^0} \{f(q-1;p-x_q) + \Delta(q;x_q) - w(c;p-x_q) + w(c;p)\}.$$

In brief, for each $p \in [t]^0$ and each $q \in \{2,3,\ldots,j\}$, we obtain

$$f(q;p) = \max_{r \in [p]^0} \{f(q-1;p-r) + \Delta(q;r) - w(c;p-r) + w(c;p)\}. \tag{9}$$

Now we describe a dynamic programming algorithm to compute $\alpha_{\vec{w}}^{(t)}(Q_c;c;p)$ for $p \in [t]^1$, which has two phases. In the first phase it computes $\Delta(i;p)$ for each $p \in [t]^0$ and each $i \in [j]^1$ by equation (4), which takes $O(j(t+1))$ time. In the second phase, for each $p \in [t]^1$, starting from $q=1$, we can take advantages of equations (8) and (9) to iteratively calculate $f(q;p)$ in increasing order of q. Until $q=j$ has been dealt with, we obtain $\alpha_{\vec{w}}^{(t)}(Q_c;c;p)$ by equation (7). It is not difficult to check that we can compute $\{\alpha_{\vec{w}}^{(t)}(Q_c;c;p) : p \in [t]^-\}$ within $O(j(t+1)^2) = O(d_G(c)(t+1)^2)$ time. Hence the lemma is true. □

We are now ready to write out explicitly the dynamic programming algorithm for the MWtSSP on vector-weighted outerplane graphs. Given a connected outerplane graph $(G,\vec{w})$ and an edge $a^*b^* \in \partial(f_0)$. We can obtain its block tree T_{B_r} in linear time by performing a block decomposition on G and then choosing a block B_r with $a^*b^* \in E(B_r)$ as root. And then we get the vertex subsets $L_0, L_1, \ldots, L_{2l}$ of T_{B_r} in linear time by executing a BFS on T_{B_r}. For convenience, the constructions of T_{B_r} and $L_0, L_1, \ldots, L_{2l}$ will be omitted in our algorithm.

In our implementation, we compute $\{\alpha_{\vec{w}}^{(t)}(R_B;par(B);p) : p \in [t]^-\}$ for each $B \in L_{2i}$ according to Claim 1 and Lemma 4.1 and $\{\alpha_{\vec{w}}^{(t)}(Q_c;c;p) : p \in [t]^-\}$ for each $c \in L_{2i-1}$ according to Lemma 4.2 for i from l to 1. The reader should note that we need to update the vector weight of c for each $c \in L_{2i-1}$ after calculating $\{\alpha_{\vec{w}}^{(t)}(Q_c;c;p) : p \in [t]^-\}$, and the specific update method is described in the proof of Lemma 4.1. After $i=1$ has been considered, we can deal with the block B_r by following similar steps to the previous blocks. At last $\alpha_{\vec{w}}^{(t)}(G)$ can be obtained by equation (1). The algorithm is presented below.

By combining the results of this section, we have the following theorem.

Theorem 4.3. *For any nonnegative integer t, the maximum weight t-sparse set problem is solvable in $O((t+2)^4 n)$ time on vector-weighted outerplane graphs.*

Proof. It can be easily seen that Theorem 3.2, Claim 1, Lemmas 4.1 and 4.2 ensure the correctness of Algorithm 3. Now we discuss the time complexity of our algorithm.

For each $i \in [l]^1$, Claim 1 and Lemma 4.1 imply that $\{\alpha_{\vec{w}}^{(t)}(R_B;par(B);p) : p \in [t]^-\}$ can be calculated in $O(|V(B)|(t+2)^4)$ time for every $B \in L_{2i}$ while Lemma 4.2 implies that the calculations of $\{\alpha_{\vec{w}}^{(t)}(Q_c;c;p) : p \in [t]^-\}$ take $O(d_G(c)(t+1)^2)$ time. Hence, the total time of all iterations is $O(\sum_{B \in V(T_{B_r})} |V(B)|(t+2)^4) + O(\sum_{c \in V(G)} d_G(c)(t+1)^2) = O(n(t+2)^4 + 2m(t+1)^2)$. Recall that $m \le 2n$ for outerplane graphs. Thus, all steps of the algorithm can be executed in $O((t+2)^4 n)$ time and the theorem follows. □

Theorem 4.3 immediately implies the following fact.

Corollary 4.4. *For any outerplane graph, both the maximum weight t-sparse set problem and the maximum t-sparse set problem can be solved in $O((t+2)^4 n)$ time. In particular, both the maximum weight independent set problem and the maximum independent set problem are solvable in linear time.*

Algorithm 3. The MWtSSP for connected vector-weighted outerplane graphs.

Input: a nonnegative integer t, a connected vector-weighted outerplane graph $(G, \vec{w})$ and its block tree T_{B_r} with the vertex subsets $L_0, L_1, \ldots, L_{2l}$ and $a^*b^* \in \partial(f_0) \cap E(B_r)$;

Output: $\alpha^{(t)}_{\vec{w}}(G)$;

for $i : l \to 1$ **do**
  for $B \in L_{2i}$ **do**
    if $B \neq K_2$ **then**
      Call Algorithm 1 on $(B, \vec{w})$, obtain $\{\alpha^{(t)}_{\vec{w}}(B; par(B); p) : p \in [t]^-\}$
    else (Suppose that $V(B) = \{c, par(B)\}$)
      Call Algorithm 2 on $(B, \vec{w})$, obtain $\{\alpha^{(t)}_{\vec{w}}(B; par(B); p) : p \in [t]^-\}$
    end if
    Set $\{\alpha^{(t)}_{\vec{w}}(R_B; par(B); p) : p \in [t]^-\} := \{\alpha^{(t)}_{\vec{w}}(B; par(B); p) : p \in [t]^-\}$
  end for

  for $c \in L_{2i-1}$ **do**
    Compute $\{\alpha^{(t)}_{\vec{w}}(Q_c; c; p) : p \in [t]^-\}$ according to Lemma 4.2
    for $p : 0 \to t$ **do**
      Set $w^*(c; p) := \max_{q \in [t-p]^0} \{\alpha^{(t)}_{\vec{w}}(Q_c; c; q) - w(c; q) + w(c; p+q)\}$
    end for
    Set $\vec{w}(c) := (\alpha^{(t)}_{\vec{w}}(Q_c; c; -1), w^*(c; 0), \ldots, w^*(c; t))$
  end for
end for

if $B_r \neq K_2$ **then**
  Call Algorithm 1 on $(B_r, \vec{w})$, obtain $\{\alpha^{(t)}_{\vec{w}}(B_r; a^*; p) : p \in [t]^-\}$
else (That is to say, $V(B_r) = \{a^*, b^*\}$)
  Call Algorithm 2 on $(B_r, \vec{w})$, obtain $\{\alpha^{(t)}_{\vec{w}}(B_r; a^*; p) : p \in [t]^-\}$
end if

Output $\alpha^{(t)}_{\vec{w}}(G) := \max\{\alpha^{(t)}_{\vec{w}}(B_r; a^*; p) : p \in [t]^-\}$

5. The MWtSSP for graphs with bounded treewidth

It is well known that the *treewidth* of an outerplane graph is at most 2. Thus it is natural to consider the MWtSSP on vector-weighted graphs with bounded treewidth. In this section, we design the dynamic programming algorithm for the MWtSSP on vector-weighted graphs with bounded treewidth. Let us begin by reviewing some relevant definitions.

Definition 5.1 ([23]). A tree decomposition of a graph G is a pair $(T, \{X_i : i \in V(T)\})$, where T is a tree whose every node i is assigned a subset $X_i \subseteq V(G)$ called a bag, such that:

(1) $\cup_{i \in V(T)} X_i = V(G)$;
(2) For any edge $uv \in E(G)$, there is an $i \in V(T)$ such that $\{u, v\} \subseteq X_i$;
(3) For any three different nodes $i, j, l \in V(T)$, if j lies on the path between i and l in T, then $X_i \cap X_l \subseteq X_j$.

The *width* of $(T, \{X_i : i \in V(T)\})$ equals $\max\{|X_i| : i \in V(T)\} - 1$. The *treewidth* of G is the minimum k such that G has a tree decomposition of width k.

Definition 5.2 ([23]). A tree decomposition $(T, \{X_i : i \in V(T)\})$ of a graph G is nice if each non-leaf node of T is of one of the following three types:

($T1$) **Introduce node**: a node i with exactly one child j such that $X_i = X_j \cup \{v\}$ for some $v \notin X_j$;
($T2$) **Forget node**: a node i with exactly one child j such that $X_i = X_j \setminus \{v\}$ for some $v \in X_j$;
($T3$) **Join node**: a node i with two children j, l such that $X_i = X_j = X_l$.

Given a graph G of treewidth k. It is well known that one can find a nice tree decomposition of width k and $O(n)$ nodes in linear time, where n is the number of vertices in G.

Suppose that $(T, \{X_i : i \in V(T)\})$ is a nice tree decomposition of G. We can assume that T is rooted at some node r. For a node i of T, let T_i be the subtree of T consisting of i and all its descendants, and let $V_i = \cup_{j \in V(T_i)} X_j$ and $G_i = G[V_i]$. We denote the number of vertices in X_i by x_i and the vertices in X_i by $v_{i_1}, v_{i_2}, \ldots, v_{i_{x_i}}$. A vector $\vec{p}_i = (p_{i_1}, p_{i_2}, \ldots, p_{i_{x_i}})$ is associated with X_i if $p_{i_j} \in [t]^-$ for each $j \in [x_i]^1$. And let

$$\mathbb{S}_{(G_i;\vec{p}_i)} = \{S : S \subseteq V_i, \Delta(G_i[S]) \leq t \text{ and } d_S(G_i; v_{i_j}) = p_{i_j} \text{ for each } v_{i_j} \in X_i\}$$

and

$$\alpha^{(t)}_{\vec{w}}(G_i;\vec{p}_i) = \begin{cases} \max\{\vec{w}(S, G_i) : S \in \mathbb{S}_{(G_i;\vec{p}_i)}\}, & \text{if } \mathbb{S}_{(G_i;\vec{p}_i)} \neq \emptyset, \\ -M, & \text{otherwise.} \end{cases}$$

Denote by $\mathcal{P}_i$ the set of the vectors $\vec{p}_i$ associated with X_i such that $\mathbb{S}_{(G_i;\vec{p}_i)} \neq \emptyset$. It is clear that $|\mathcal{P}_i| \leq (t+2)^{x_i}$ for any $i \in V(T)$. Due to $V_r = V(G)$, we have $\alpha^{(t)}_{\vec{w}}(G) = \max\{\alpha^{(t)}_{\vec{w}}(G_r;\vec{p}_r) : \vec{p}_r \in \mathcal{P}_r\}$. We are now prepared to state our algorithms.

Lemma 5.3. *Suppose i is a leaf of T. Then one can compute $\mathcal{P}_i$ and $\{\alpha^{(t)}_{\vec{w}}(G_i;\vec{p}_i) : \vec{p}_i \in \mathcal{P}_i\}$ within $O((k+1)2^{k+1})$ time.*

Proof. It is clear that $\mathcal{P}_i = \{(d_S(G_i; v_{i_1}), d_S(G_i; v_{i_2}), \ldots, d_S(G_i; v_{i_{x_i}})) : S \subseteq X_i \text{ and } \Delta(G_i[S]) \leq t\}$ and $\alpha^{(t)}_{\vec{w}}(G_i;\vec{p}_i) = \sum_{v \in X_i} w(v; d_S(G_i; v))$ for each $\vec{p}_i \in \mathcal{P}_i$. Notice that $|X_i| \leq k+1$, $|\mathcal{P}_i| \leq 2^{k+1}$. Thus one can compute $\mathcal{P}_i$ and $\{\alpha^{(t)}_{\vec{w}}(G_i;\vec{p}_i) : \vec{p}_i \in \mathcal{P}_i\}$ within $O((k+1)2^{k+1})$ time. □

We now move on to presenting how to compute $\mathcal{P}_i$ and $\{\alpha^{(t)}_{\vec{w}}(G_i;\vec{p}_i) : \vec{p}_i \in \mathcal{P}_i\}$ for a non-leaf node i, when $\mathcal{P}_j$ and $\{\alpha^{(t)}_{\vec{w}}(G_j;\vec{p}_j) : \vec{p}_j \in \mathcal{P}_j\}$ are known for each child j of i.

Introduce node. Suppose i is an introduce node of T. Let j be the unique child of i. Before we state our algorithm, we make a useful observation.

Observation 5.4. Let i be an introduce node of T with the child j and $X_i = X_j \cup \{v\}$. Then $v \notin V_j$ and $N_G(v) \cap V_i \subseteq X_i$.

Proof. First we prove that $v \notin V_j$. If not, then $v \in X_l$ for some $l \in V(T_j)$ and $l \neq j$. Because j is the only child of i in T, j lies on the path between i and l in T, which is a contradiction to Definition 5.1(3) since $v \notin X_j$. Therefore, $v \notin V_j$.

Then we prove that $N_G(v) \cap V_i \subseteq X_i$. If there is a vertex $u \in (N_G(v) \cap V_i) \setminus X_i$, then $u \in X_l$ for some $l \in V(T_i)$ and $l \notin \{i, j\}$. According to Definition 5.1(2), there is a bag $X_{l'}$ such that $\{u, v\} \subseteq X_{l'}$. Since $v \notin V_j$ and $u \notin X_i$, $l' \in V(T) \setminus V(T_i)$. Therefore, i lies on the path between l and l' in T, contradicting the fact that $u \notin X_i$. This finishes the proof. □

We may assume that $X_i = X_j \cup \{v_{i_{x_i}}\}$ and $v_{i_q} = v_{j_q}$ for each $q \in [x_j]^1$. And let $I_i := \{q : v_{i_{x_i}} v_{i_q} \in E(G) \text{ and } q \in [x_j]^1\}$. Notice that if S is a t-sparse set of G_i, then $S \setminus \{v_{i_{x_i}}\}$ must be a t-sparse set of G_j. Now we describe the algorithm.

Lemma 5.5. *Suppose i is an introduce node of T with a child j. If $\mathcal{P}_j$ and $\{\alpha^{(t)}_{\vec{w}}(G_j;\vec{p}_j) : \vec{p}_j \in \mathcal{P}_j\}$ are known, then $\mathcal{P}_i$ and $\{\alpha^{(t)}_{\vec{w}}(G_i;\vec{p}_i) : \vec{p}_i \in \mathcal{P}_i\}$ can be calculated in $O(k(t+2)^k)$ time through Algorithm 4.*

Proof. The correctness of Algorithm 4 follows from Observation 5.4. Because $|X_j| \leq k$, each iteration can be implemented in $O(k)$ time. Notice that $|\mathcal{P}_j| \leq (t+2)^k$, the total running time is $O(k(t+2)^k)$. □

Algorithm 4. Introduce node.

Input: $\mathcal{P}_j$ and $\{\alpha_{\vec{w}}^{(t)}(G_j;\vec{p}_j):\vec{p}_j\in\mathcal{P}_j\}$;
Output: $\mathcal{P}_i$ and $\{\alpha_{\vec{w}}^{(t)}(G_i;\vec{p}_i):\vec{p}_i\in\mathcal{P}_i\}$;
Set $\mathcal{P}_i:=\emptyset$
for $\vec{p}_j\in\mathcal{P}_j$ **do**
  Set $\mathcal{P}_i:=\mathcal{P}_i\cup\{(p_{j_1},p_{j_2},\dots,p_{j_{x_j}},-1)\}$
  Set $\alpha_{\vec{w}}^{(t)}(G_i;(p_{j_1},p_{j_2},\dots,p_{j_{x_j}},-1)):=\alpha_{\vec{w}}^{(t)}(G_j;\vec{p}_j)+w(v_{x_i};-1)$
  if $p_{j_q}\le t-1$ for each $q\in I_i$ **then**
    Set $p_{i_{x_i}}:=0$ and $\Delta:=0$
    for $q:1\to x_j$ **do**
      if $q\in I_i$ and $p_{j_q}\ge 0$ **then**
        Set $p_{i_{x_i}}:=p_{i_{x_i}}+1$, $p_{i_q}:=p_{j_q}+1$ and $\Delta:=\Delta-w(v_{i_q};p_{j_q})+w(v_{i_q};p_{i_q})$
      else
        Set $p_{i_q}:=p_{j_q}$
      end if
    end for
    if $0\le p_{i_{x_i}}\le t$ **then**
      Set $\mathcal{P}_i:=\mathcal{P}_i\cup\{(p_{i_1},p_{i_2},\dots,p_{i_{x_i}})\}$
      Set $\alpha_{\vec{w}}^{(t)}(G_i;(p_{i_1},p_{i_2},\dots,p_{i_{x_i}})):=\alpha_{\vec{w}}^{(t)}(G_j;\vec{p}_j)+\Delta+w(v_{x_i};p_{i_{x_i}})$
    end if
  end if
end for
Output $\mathcal{P}_i$ and $\{\alpha_{\vec{w}}^{(t)}(G_i;\vec{p}_i):\vec{p}_i\in\mathcal{P}_i\}$

Algorithm 5. Forget node.

Input: $\mathcal{P}_j$ and $\{\alpha_{\vec{w}}^{(t)}(G_j;\vec{p}_j):\vec{p}_j\in\mathcal{P}_j\}$;
Output: $\mathcal{P}_i$ and $\{\alpha_{\vec{w}}^{(t)}(G_i;\vec{p}_i):\vec{p}_i\in\mathcal{P}_i\}$;
Set $\mathcal{P}_i:=\emptyset$
for $\vec{p}_j\in\mathcal{P}_j$ **do**
  Set $\vec{p}_i:=(p_{j_1},p_{j_2},\dots,p_{j_{x_j-1}})$
  if $\vec{p}_i\notin\mathcal{P}_i$ **then**
    Set $\mathcal{P}_i:=\mathcal{P}_i\cup\{\vec{p}_i\}$ and $\alpha_{\vec{w}}^{(t)}(G_i;\vec{p}_i):=\alpha_{\vec{w}}^{(t)}(G_j;\vec{p}_j)$
  else if $\alpha_{\vec{w}}^{(t)}(G_j;\vec{p}_j)>\alpha_{\vec{w}}^{(t)}(G_i;\vec{p}_i)$ **then**
    Set $\alpha_{\vec{w}}^{(t)}(G_i;\vec{p}_i):=\alpha_{\vec{w}}^{(t)}(G_j;\vec{p}_j)$
  end if
end for
Output $\mathcal{P}_i$ and $\{\alpha_{\vec{w}}^{(t)}(G_i;\vec{p}_i):\vec{p}_i\in\mathcal{P}_i\}$

Forget node. Suppose i is a forget node of T with the child j. We assume that $X_i=X_j\setminus\{v_{i_{x_j}}\}$ and $v_{i_q}=v_{j_q}$ for each $q\in[x_i]$[1]. It is clear that $V_i=V_j$ and $G_i=G_j$. The algorithm is presented below.

Lemma 5.6. *Suppose i is a forget node of T with a child j. If $\mathcal{P}_j$ and $\{\alpha_{\vec{w}}^{(t)}(G_j;\vec{p}_j):\vec{p}_j\in\mathcal{P}_j\}$ are known, then $\mathcal{P}_i$ and $\{\alpha_{\vec{w}}^{(t)}(G_i;\vec{p}_i):\vec{p}_i\in\mathcal{P}_i\}$ can be calculated in $O(k(t+2)^{k+1})$ time through Algorithm 5.*

Proof. According to Definition 5.1(3), we have $v_{i_{x_j}}\notin X_l$ for any $l\in V(T)\setminus V(T_j)$ as otherwise there is a contradiction to the fact that $v_{i_{x_j}}\notin X_i$. Thus Algorithm 5 is clearly correct. For each $\vec{p}_j\in\mathcal{P}_j$, it takes $O(k)$ time to obtain $\vec{p}_i$ and check whether $\vec{p}_i$ is in $\mathcal{P}_i$. Hence, each iteration takes $O(k)$ time. Since $|\mathcal{P}_j|\le(t+2)^{k+1}$, Algorithm 5 needs $O(k(t+2)^{k+1})$ time. □

Join node. Suppose i is a join node of T with two children j and l. It is clear that $X_i = X_j = X_l$, $V_j \cap V_l = X_i$ and $G_i = G_j \cup G_l$. We may assume that $v_{i_q} = v_{j_q} = v_{l_q}$ for each $q \in [x_i]^1$. In order to describe the algorithm more easily, we need some other definitions.

Let z be a node of T. Given a vector $\vec{p}_z \in \mathcal{P}_z$, for each $q \in [x_z]^1$, we define

$$d_{z_q} = \begin{cases} |\{v_{z_y} : v_{z_y} \in X_z, v_{z_q}v_{z_y} \in E(G) \text{ and } p_{z_y} \geq 0\}|, & \text{if } p_{z_q} \geq 0, \\ -1, & \text{otherwise.} \end{cases}$$

We call $\vec{d_z} = (d_{z_1}, d_{z_2}, \ldots, d_{z_{x_z}})$ the vector with respect to $\vec{p}_z$.

Notice that if a vertex subset S is a t-sparse set of G_i, then $S \cap V_j$ (resp. $S \cap V_l$) must be a t-sparse set of G_j (resp. G_l). We immediately observe the following.

Observation 5.7. Let i be a join node of T with two children j and l. For any two vectors $\vec{p}_j \in \mathcal{P}_j$ and $\vec{p}_l \in \mathcal{P}_l$, if $p_{j_q} = p_{l_q} = -1$ or $p_{j_q} \geq 0$ and $p_{l_q} \geq 0$ for each $q \in [x_i]^1$, then $\vec{d_j} = \vec{d_l}$.

Based on the above observation, we proceed to introduce the following notion.

Definition 5.8. Let i be a join node of T with two children j and l. For any two vectors $\vec{p}_j \in \mathcal{P}_j$ and $\vec{p}_l \in \mathcal{P}_l$, we say that they are combinable if the following two properties hold:

(1) for each $q \in [x_i]^1$, $p_{j_q} = p_{l_q} = -1$ or $p_{j_q} \geq 0$ and $p_{l_q} \geq 0$;
(2) for each $q \in [x_i]^1$, $p_{j_q} + p_{l_q} - d_{j_q} \leq t$.

Now we present the algorithm.

Algorithm 6. Join node.

Input: $\mathcal{P}_j$, $\{\alpha_{\vec{w}}^{(t)}(G_j; \vec{p}_j) : \vec{p}_j \in \mathcal{P}_j\}$, $\mathcal{P}_l$ and $\{\alpha_{\vec{w}}^{(t)}(G_l; \vec{p}_l) : \vec{p}_l \in \mathcal{P}_l\}$;
Output: $\mathcal{P}_i$ and $\{\alpha_{\vec{w}}^{(t)}(G_i; \vec{p}_i) : \vec{p}_i \in \mathcal{P}_i\}$;

Set $\mathcal{P}_i := \emptyset$
for $\vec{p}_j \in \mathcal{P}_j$ **do**
  for $\vec{p}_l \in \mathcal{P}_l$ **do**
    if $\vec{p}_j$ and $\vec{p}_l$ are combinable **then**
      Set $\Delta := 0$ and $W := 0$
      for $q : 1 \to x_i$ **do**
        if $p_{j_q} = -1$ **then**
          Set $p_{i_q} := -1$ and $\Delta := \Delta - w(v_{i_q}; -1)$
        else
          Set $p_{i_q} := p_{j_q} + p_{l_q} - d_{j_q}$ and $\Delta := \Delta - w(v_{i_q}; p_{j_q}) - w(v_{i_q}; p_{l_q}) + w(v_{i_q}; p_{i_q})$
        end if
      end for
      Set $\vec{p}_i := (p_{i_1}, p_{i_2}, \ldots, p_{i_{x_i}})$ and $W := \alpha_{\vec{w}}^{(t)}(G_j; \vec{p}_j) + \alpha_{\vec{w}}^{(t)}(G_l; \vec{p}_l) + \Delta$
      if $\vec{p}_i \notin \mathcal{P}_i$ **then**
        Set $\mathcal{P}_i := \mathcal{P}_i \cup \{\vec{p}_i\}$ and $\alpha_{\vec{w}}^{(t)}(G_i; \vec{p}_i) := W$
      else if $W > \alpha_{\vec{w}}^{(t)}(G_i; \vec{p}_i)$ **then**
        Set $\alpha_{\vec{w}}^{(t)}(G_i; \vec{p}_i) := W$
      end if
    end if
  end for
end for
Output $\mathcal{P}_i$ and $\{\alpha_{\vec{w}}^{(t)}(G_i; \vec{p}_i) : \vec{p}_i \in \mathcal{P}_i\}$

Lemma 5.9. *Suppose i is a join node of T with two children j and l. If $\mathcal{P}_z$ and $\{\alpha_{\vec{w}}^{(t)}(G_z;\vec{p}_z) : \vec{p}_z \in \mathcal{P}_z\}$ are known for each $z \in \{j,l\}$, then $\mathcal{P}_i$ and $\{\alpha_{\vec{w}}^{(t)}(G_i;\vec{p}_i) : \vec{p}_i \in \mathcal{P}_i\}$ can be calculated in $O((k+1)(t+2)^{2k+2})$ time through Algorithm 6.*

Proof. Algorithm 6 is obviously true. Given two vectors $\vec{p}_j \in \mathcal{P}_j$ and $\vec{p}_l \in \mathcal{P}_l$, due to Definition 5.8, it takes $O(k+1)$ time to check whether they are combinable. While they are combinable, it takes $O(k+1)$ time to update $\mathcal{P}_i$ and $\{\alpha_{\vec{w}}^{(t)}(G_i;\vec{p}_i) : \vec{p}_i \in \mathcal{P}_i\}$. Recall that $|\mathcal{P}_j| \leq (t+2)^{k+1}$ and $|\mathcal{P}_l| \leq (t+2)^{k+1}$, the total time is $O((k+1)(t+2)^{2k+2})$. □

Finally, the dynamic programming algorithm for the MWtSSP on graphs with bounded treewidth is presented below.

Algorithm 7. The MWtSSP for graphs with bounded treewidth.

Input: a vector-weighted graph $(G,\vec{w})$ and its nice tree decomposition $(T,\{X_i : i \in V(T)\})$;
Output: $\alpha_{\vec{w}}^{(t)}(G)$;

for each leaf i of T **do**
  Compute $\mathcal{P}_i$ and $\{\alpha_{\vec{w}}^{(t)}(G_i;\vec{p}_i) : \vec{p}_i \in \mathcal{P}_i\}$ according to Lemma 5.3
  color the node i red
end for

while there is a node i of T that is uncolored but all its children are colored **do**
  if i is an introduce node **then**
    Call Algorithm 4 to compute $\mathcal{P}_i$ and $\{\alpha_{\vec{w}}^{(t)}(G_i;\vec{p}_i) : \vec{p}_i \in \mathcal{P}_i\}$
  else if i is a forget node **then**
    Call Algorithm 5 to compute $\mathcal{P}_i$ and $\{\alpha_{\vec{w}}^{(t)}(G_i;\vec{p}_i) : \vec{p}_i \in \mathcal{P}_i\}$
  else
    Call Algorithm 6 to compute $\mathcal{P}_i$ and $\{\alpha_{\vec{w}}^{(t)}(G_i;\vec{p}_i) : \vec{p}_i \in \mathcal{P}_i\}$
  end if
  color the node i red
end while
Output $\alpha_{\vec{w}}^{(t)}(G) = \max\{\alpha_{\vec{w}}^{(t)}(G_r;\vec{p}_r) : \vec{p}_r \in \mathcal{P}_r\}$.

Combining all the results of this section, we have the following theorem.

Theorem 5.10. *For any nonnegative integer t, the maximum weight t-sparse set problem is solvable in $O((k+1)(t+2)^{2k+2}n)$ time on vector-weighted graphs of treewidth k.*

Proof. Lemmas 5.3, 5.5, 5.6 and 5.9 ensure the correctness of Algorithm 7. Recall that a nice tree decomposition of a graph has $O(n)$ nodes, the time complexity of Algorithm 7 is $O((k+1)(t+2)^{2k+2}n)$. Thus the theorem is proved. □

Theorem 5.10 implies the following fact.

Corollary 5.11. *For any graph of treewidth k, both the maximum weight t-sparse set problem and the maximum t-sparse set problem can be solved in $O((k+1)(t+2)^{2k+2}n)$ time.*

6. Summary

On the theoretical side, this paper offered a straightforward generalization of the traditional maximum weight independent set problem. We defined vector weights on graphs and proposed the maximum weight t-sparse set

problem (MWtSSP) on vector-weighted graphs. We also derived two dynamic programming algorithms for this problem on outerplane graphs and graphs with bounded treewidth, respectively.

It should be pointed out that both Algorithms 3 and 7 are applicable to outerplane graphs. However, the time complexity of the latter is $O((t+2)^6 n)$ while the time complexity of the former is only $O((t+2)^4 n)$.

A natural future research direction is to consider this problem on some other specific classes of graphs, such as claw-free graphs, planar graphs, and so on. Moreover, it would be very significant to consider whether parameterization methods can be developed to solve the MWtSSP for general vector-weighted graphs.

Furthermore, it is quite interesting to study a further generalization of MWtSSP on vector-weighted graphs, which is presented as follows. Given a positive integer k, a nonnegative integer t and a vector-weighted graph $(G, \vec{w}, t)$, find k disjoint t-sparse sets $S_1, S_2, \ldots, S_k$ in $(G, \vec{w}, t)$ such that $\sum_{i=1}^{k} w(S_i, G)$ is maximum.

Finally, another avenue for future research is to study some special coloring problems on vector-weighted graphs. For example, given a fixed positive integer p, a nonnegative integer t and a vector-weighted graph $(G, \vec{w}, t)$, how do we color $(G, \vec{w}, t)$ with the fewest colors so that each color class is a t-sparse set of weight at most p?

Funding. This work was supported by by NSFC 11771080.

Acknowledgements. We would like to sincerely thank the anonymous reviewers for their careful reading and valuable suggestions to improve this paper.

References

[1] B. Balasundaram, S.S. Chandramouli and S.Trukhanov, Approximation algorithms for finding and partitioning unit-disk graphs into co-k-plexes. *Optim. Lett.* **4** (2010) 311–320.

[2] B. Balasundaram, S. Butenko and I.V. Hicks, Clique relaxations in social network analysis: the maximum k-plex problem. *Oper. Res.* **59** (2011) 133–142.

[3] S. Bandyapadhyay, A variant of the maximum weight independent set problem. Preprint arXiv:1409.0173 (2014).

[4] N. Betzler, R. Bredereck, R. Niedermeier and J. Uhlmann, On bounded-degree vertex deletion parameterized by treewidth. *Discrete Appl. Math.* **160** (2012) 53–60.

[5] A.A. Bhave, *Greedy randomized adaptive search procedure for the maximum co-k-plex problem.* Ph.D. Thesis, Oklahoma State University (2010).

[6] A. Brandstädt, V.V. Lozin and R. Mosca, Independent sets of maximum weight in apple-free graphs. *SIAM J. Discrete Math.* **24** (2010) 239–254.

[7] S. Carlsson, Y. Igarashi, K. Kanai, A. Lingas, K. Miura and O. Petersson, Information disseminating schemes for fault tolerance in hypercubes. *IEICE Trans. Fundam.* **75** (1992) 255–260.

[8] G.H. Chen, M.T. Kuo and J.P. Sheu, An optimal time algorithm for finding a maximum weight independent set in a tree. *BIT* **28** (1988) 353–356.

[9] M. Cygan, F.V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk and S. Saurabh, Parameterized Algorithms. Springer (2015) 162–167.

[10] A. Dessmark, K. Jansen and A. Lingas, The maximum k-dependent and f-dependent set problem, in International Symposium on Algorithms and Computation. Springer (1993) 88–97.

[11] H. Djidjev, O. Garrido, C. Levcopoulos and A. Lingas, On the maximum k-dependent set problem. Technical Report LU-CS-TR:92-91, Department of Computer Science, Lund University, Sweden (1992).

[12] T. Ekim and J. Gimbel, Some defective parameters in graphs. *Graphs Comb.* **29** (2013) 213–224.

[13] M.R. Fellows, J. Guo, H. Moser and R. Niedermeier, A generalization of Nemhauser and Trotter's local optimization theorem. *J. Comput. Syst.* **77** (2011) 1141–1158.

[14] H. Fichtenberger, P. Peng and C. Sohler, Every testable (infinite) property of bounded-degree graphs contains an infinite hyperfinite subproperty, in Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms. SIAM (2019) 714–726.

[15] N. Fountoulakis, R.J. Kang and C. McDiarmid, The t-stability number of a random graph. *Electron. J. Comb.* **17** (2010) R59.

[16] S. Hosseinian and S. Butenko, An improved approximation for maximum k-dependent set on bipartite graphs. *Discrete Appl. Math.* **307** (2022) 95–101.

[17] F. Havet, R.J. Kang and J. Sereni, Improper coloring of unit disk graphs. *Networks* **54** (2009) 150–164.

[18] M. Kumar, S. Mishra, N.S. Devi and S. Saurabh, Approximation algorithms for node deletion problems on bipartite graphs with finite forbidden subgraph characterization. *Theor. Comput. Sci.* **526** (2014) 90–96.

[19] V.V. Lozin and M. Milanič, A polynomial algorithm to find an independent set of maximum weight in a fork-free graph. *J. Discrete Algorithms* **6** (2008) 595–604.

[20] G.J. Minty, On maximal independent sets of vertices in claw-free graphs. *J. Comb. Theory Ser. B* **28** (3) (1980) 284–304.
[21] B. McClosky, J.D. Arellano and I.V. Hicks, Co-2-plex vertex partitions. *J. Comb. Optim.* **30** (2015) 729–746.
[22] I. Newman and C. Sohler, Every property of hyperfinite graphs is testable. *SIAM J. Comput.* **42** (2013) 1095–1112.
[23] R. Niedermeier, Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006) 88–89.
[24] S.B. Seidman and B.L. Foster, A graph-theoretic generalization of the clique concept. *J. Math. Soc.* **6** (1978) 139–154.
[25] J. Tu, Y. Li and J. Du, Maximal and maximum dissociation sets in general and triangle-free graphs. *Appl. Math. Comput.* **426** (2022) 127107.
[26] J. Tu, Z. Zhang and Y. Shi, The maximum number of maximum dissociation sets in trees. *J. Graph Theory* **96** (2020) 472–489.
[27] M. Xiao and S. Kou, Exact algorithms for the maximum dissociation set and minimum 3-path vertex cover problems. *Theor. Comput. Sci.* **657** (2017) 86–97.
[28] M. Yannakakis, Node-deletion problems on bipartite graphs. *SIAM J. Comput.* **10** (1981) 310–327.

Please help to maintain this journal in open access!

This journal is currently published in open access under the Subscribe to Open model (S2O). We are thankful to our subscribers and supporters for making it possible to publish this journal in open access in the current year, free of charge for authors and readers.

Check with your library that it subscribes to the journal, or consider making a personal donation to the S2O programme by contacting subscribers@edpsciences.org.

More information, including a list of supporters and financial transparency reports, is available at https://edpsciences.org/en/subscribe-to-open-s2o.