The Minimum Vulnerability Problem on Graphs

Yusuke Aoki[1], Bjarni V. Halldórsson[2], Magnús M. Halldórsson[2⋆], Takehiro Ito[1⋆⋆], Christian Konrad[2], and Xiao Zhou[1]

[1] Graduate School of Information Sciences, Tohoku University, Japan.
{y.aoki, takehiro, zhou}@ecei.tohoku.ac.jp
[2] School of Computer Science, Reykjavík University, Iceland.
{mmh, bjarnivh, christiank}@ru.is

Abstract. Suppose that each edge e of an undirected graph G is associated with three nonnegative integers $\mathsf{cost}(e)$, $\mathsf{vul}(e)$ and $\mathsf{cap}(e)$, called the cost, vulnerability and capacity of e, respectively. Then, we consider the problem of finding k paths in G between two prescribed vertices with the minimum total cost; each edge e can be shared without cost by at most $\mathsf{vul}(e)$ paths, and can be shared by more than $\mathsf{vul}(e)$ paths if we pay $\mathsf{cost}(e)$, but cannot be shared by more than $\mathsf{cap}(e)$ paths even if we pay the cost of e. This problem generalizes the disjoint path problem, the minimum shared edges problem and the minimum edge cost flow problem for undirected graphs, and it is known to be NP-hard. In this paper, we study the problem from the viewpoint of specific graph classes, and give three results. We first show that the problem remains NP-hard even for bipartite series-parallel graphs and for threshold graphs. We then give a pseudo-polynomial-time algorithm for bounded treewidth graphs. Finally, we give a fixed-parameter algorithm for chordal graphs when parameterized by the number k of required paths.

1 Introduction

In this paper, we study the minimum vulnerability problem on undirected graphs, originally introduced by Assadi et al. [1]. This problem has strong relationships to several well-known problems such as the disjoint path problem [6], the minimum shared edges problem [1, 9, 10] and the minimum edge cost flow problem [6, 8]. It is defined as follows.

Let $G = (V, E)$ be an undirected and connected graph; we sometimes denote by $V(G)$ and $E(G)$ the vertex set and edge set of G, respectively. Suppose that each edge $e \in E(G)$ is associated with three nonnegative integers $\mathsf{cost}(e)$, $\mathsf{vul}(e)$ and $\mathsf{cap}(e)$, called the *cost*, *vulnerability* and *capacity* of e, respectively. (See Fig. 1(a) as an example.) Let $\mathcal{P}$ be a multi-set of paths in G. Then, for each

⋆ Magnús M. Halldórsson and Christian Konrad are supported by Icelandic Research Fund grant-of-excellence no. 120032011.
⋆⋆ This work is partially supported by JSPS KAKENHI 25106504 and 25330003.

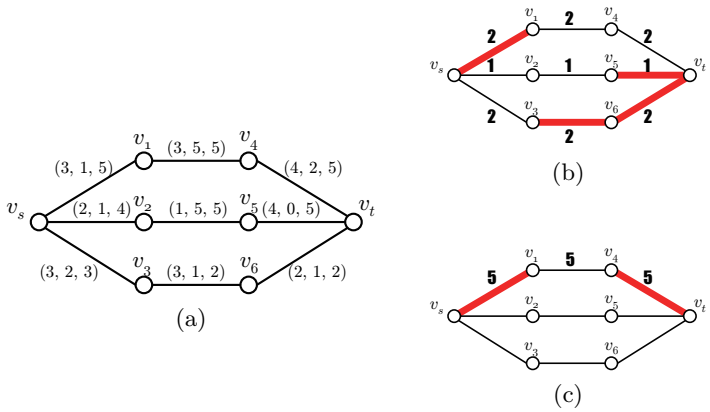

Fig. 1. (a) An instance for the minimum vulnerability problem, where the triple attached to each edge e represents the three weights $(\mathsf{cost}(e), \mathsf{vul}(e), \mathsf{cap}(e))$. (b) A feasible solution for $k = 5$ such that $\lambda(\mathcal{P}) = 12$, where the bold number attached to each edge e represents $\mu(e, \mathcal{P})$ and we have to pay the cost for the (red) thick edges. (c) An optimal solution $\mathcal{P}^*$ for $k = 5$, where $\lambda(\mathcal{P}^*) = \mathsf{OPT}_5(G) = 7$.

edge $e \in E(G)$, we define $\mu(e, \mathcal{P})$ to be the number of paths in $\mathcal{P}$ that contain e, and define the *penalty* $\lambda(e, \mathcal{P})$ of e on $\mathcal{P}$ as follows:

$$\lambda(e, \mathcal{P}) = \begin{cases} 0 & \text{if } 0 \le \mu(e, \mathcal{P}) \le \mathsf{vul}(e); \\ \mathsf{cost}(e) & \text{if } \mathsf{vul}(e) < \mu(e, \mathcal{P}) \le \mathsf{cap}(e); \\ +\infty & \text{otherwise,} \end{cases}$$

that is, each edge e can be shared without cost by at most $\mathsf{vul}(e)$ paths in $\mathcal{P}$, and cannot be shared by more than $\mathsf{cap}(e)$ paths even if we pay the cost of e. Then, the *penalty* of $\mathcal{P}$, denoted by $\lambda(\mathcal{P})$, is defined as $\lambda(\mathcal{P}) = \sum_{e \in E(G)} \lambda(e, \mathcal{P})$. See Fig. 1(b) and (c) for an example.

Given an edge-weighted graph G with two specified vertices $v_s, v_t \in V(G)$ and a nonnegative integer k, the *minimum vulnerability problem* is to find a set $\mathcal{P}$ consisting of exactly k (not necessarily distinct) paths in G between v_s and v_t, called $v_s v_t$-*paths* in short, such that the penalty $\lambda(\mathcal{P})$ of $\mathcal{P}$ is minimized. We denote by $\mathsf{OPT}_k(G, v_s, v_t)$, in short by $\mathsf{OPT}_k(G)$, the optimal value of the minimum vulnerability problem for a given graph G.

This problem arises in the context of communication network design, reliable multicast communications, and distributed communication protocols [1, 9]. For example, consider the data transfer of some important data from a server v_s to another one v_t via a computer network formulated as a graph G. To avoid the attack of hackers, we first divide the data into k smaller chunks of data, and we wish to send them along disjoint paths in G. If G does not have k disjoint $v_s v_t$-paths, then some data must share edges (i.e., links in the network); assume that links use different security protocols, and hence they have different capacities, vulnerabilities and costs to be shared. Thus, the minimum vulnerability problem

formulates this situation, and an optimal solution provides a way to send the data at minimum cost while satisfying given security requirements.

1.1 Known and related results

The minimum vulnerability problem was originally defined on digraphs (i.e., directed graphs) [1], in which we wish to find k directed paths from v_s to v_t. The problem on digraphs generalizes the minimum shared edges problem [1, 9, 10] and the minimum edge cost flow problem [6, 8] as follows: (In the following, we denote by n the number of vertices in a graph G, and by m the number of edges or arcs in G.)

• The minimum shared edges problem:

This problem corresponds to the minimum vulnerability problem on digraphs restricted to the case where $\mathsf{cost}(e) = 1$, $\mathsf{vul}(e) = 1$ and $\mathsf{cap}(e) = k$ for all arcs e in a given digraph. The problem is known to be NP-hard [9], and even hard to approximate within a factor of $2^{\log^{1-\varepsilon} n}$ for any constant $\varepsilon > 0$ [9]. On the other hand, Assadi et al. [1] showed that the minimum shared edges problem can be approximated in polynomial time within a factor of $\min\{n^{\frac{3}{4}}, m^{\frac{1}{2}}\}$, or a factor of $\lfloor k/2 \rfloor$.

Furthermore, there exists a pseudo-polynomial-time algorithm for the minimum shared edges problem when restricted to undirected graphs G with bounded treewidth [10]. Its running time is $O\Big(n(k+1)^{2^{(t(t+1)/2)}} + n(k+1)^{(t+4)^{(2t+8)}}\Big)$, where t is the treewidth of G. (The definition of treewidth will be given in Section 3.)

• The minimum edge cost flow problem:

This problem corresponds to the minimum vulnerability problem on digraphs restricted to the case where $\mathsf{vul}(e) = 0$ for all arcs e in a given digraph. This problem is known to be strongly NP-hard even for bipartite digraphs [8], and hard even to approximate within a factor of $2^{\log^{1-\varepsilon} n}$ for any constant $\varepsilon > 0$ [5].

The minimum edge cost flow problem remains NP-hard even for series-parallel digraphs [8], but it admits a fully polynomial-time approximation scheme (FPTAS) for series-parallel digraphs [8].

The minimum vulnerability problem.

We now explain known results for our problem. Since the minimum vulnerability problem on digraphs is a generalization of the minimum shared edges problem and the minimum edge cost flow problem, all hardness results obtained for the latter two problems hold for the problem on digraphs, too. Furthermore, the strong NP-hardness proof given in [8] can easily be extended to bipartite undirected graphs, and hence the minimum vulnerability problem remains strongly NP-hard even for bipartite undirected graphs.

However, this relation does not hold in the other direction, that is, algorithms obtained for the two problems above do not always work for the minimum vulnerability problem even on undirected graphs. Thus, Assadi et al. [1] developed an $O\big(n^3 m^{2(k-1)}\big)$-time algorithm which exactly solves the minimum vulnerability problem on any digraph for the case where all arcs e have identical positive

vulnerability $\mathsf{vul}(e) = r \geq 1$. They also gave an approximation result for the case where $r \geq 0$: The best known approximation ratio is $\lfloor \frac{k}{r+1} \rfloor$ for general digraphs [1]. As far as we know, these are the only positive results known for the minimum vulnerability problem on digraphs.

1.2 Our contributions

In this paper, we study the minimum vulnerability problem on undirected graphs from the viewpoint of specific graph classes, and mainly give the following three results. (We will later define the graph classes mentioned below.)

First, we show that the problem remains NP-hard for undirected graphs, more specifically, for bipartite series-parallel graphs and for threshold graphs, even if $\mathsf{cap}(e) \geq 1$ and $\mathsf{vul}(e) \geq 1$ holds for all edges e in a graph G. Therefore, it is very unlikely that the problem can be solved in polynomial time even for these very restricted graph classes. It is important that the result holds under the condition that $\mathsf{cap}(e) \geq 1$ and $\mathsf{vul}(e) \geq 1$ holds for all edges $e \in E(G)$, because otherwise any graph can be represented as a complete graph (which is a threshold graph) by appropriately choosing edge-costs.

Second, we give a pseudo-polynomial-time algorithm for bounded treewidth graphs, which form a super-class of series-parallel graphs; note that our algorithm works also for the case where $\mathsf{cap}(e) = 0$ or $\mathsf{vul}(e) = 0$ holds for some edges e. Thus, this algorithm solves the minimum shared edges problem and the minimum edge cost flow problem for undirected graphs, too. Furthermore, our algorithm improves the best running time known for the minimum shared edges problem on undirected graphs with bounded treewidth [10].

Third, by taking the number k of required $v_s v_t$-paths as a parameter, we give a fixed-parameter algorithm for chordal graphs G such that $\mathsf{vul}(e) \geq 1$ holds for all edges $e \in E(G)$. Note that the problem is NP-hard for chordal graphs, because chordal graphs form a super-class of threshold graphs.

2 Computational Hardness

In this section, we clarify the complexity status of the minimum vulnerability problem. First, we give a reduction showing that the problem is NP-hard for bipartite series-parallel graphs. We then show that this reduction can be extended to an NP-hardness proof for threshold graphs.

2.1 Bipartite series-parallel graphs

Subdividing an edge (u, v) of a graph is the operation of deleting the edge (u, v) and adding a path between u and v through several newly added vertices of degree two. A graph G is said to be a *subdivision* of a graph G' if G is obtained from G' by subdividing some of the edges of G'. A graph is *series-parallel* if it does not contain a subdivision of a complete graph K_4 on four vertices as a subgraph [4].

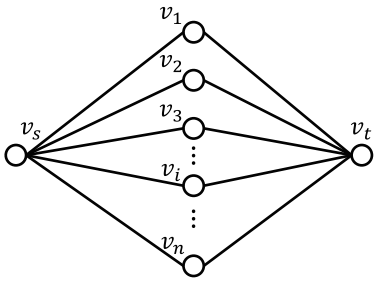

Fig. 2. A series-parallel graph used in the reduction.

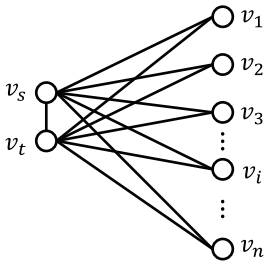

Fig. 3. A threshold graph used in the reduction.

Theorem 1. *The minimum vulnerability problem is NP-hard for bipartite series-parallel graphs, even if* $\mathsf{cap}(e) \geq 1$ *and* $\mathsf{vul}(e) = r$ *hold for all edges* $e \in E(G)$*, where* $r \geq 1$ *is any fixed constant.*

Proof. We give a polynomial-time reduction from Knapsack [6]. In an instance of Knapsack, we are given a set A of n items $a_1, a_2, \ldots, a_n$, a positive integer weight $w(a_i)$ and a positive integer profit $p(a_i)$ for each item $a_i \in A$, and two positive integers c and d. Then, the Knapsack problem is to determine whether there exists a subset $A' \subseteq A$ such that the total weight of A' is at most c and the total profit of A' is at least d. This problem is known to be NP-complete [6].

We indeed prove that the following decision version of the minimum vulnerability problem is NP-hard: Given a graph G with two specified vertices $v_s, v_t \in V(G)$ associated with three nonnegative integers $\mathsf{cost}(e)$, $\mathsf{vul}(e)$ and $\mathsf{cap}(e)$, and two nonnegative integers k and c, is there a set $\mathcal{P}$ consisting of exactly k $v_s v_t$-paths on G such that the penalty $\lambda(\mathcal{P})$ of $\mathcal{P}$ is at most c?

We first construct the corresponding graph G associated with three integers $\mathsf{cost}(e)$, $\mathsf{vul}(e)$ and $\mathsf{cap}(e)$. For each item a_i, $1 \leq i \leq n$, we add a vertex v_i to $V(G)$ corresponding to a_i. Then, we add two vertices v_s and v_t to $V(G)$, and for each i, $1 \leq i \leq n$, we add two edges (v_s, v_i) and (v_i, v_t) to $E(G)$. We set three integers $\mathsf{cost}(e)$, $\mathsf{vul}(e)$ and $\mathsf{cap}(e)$ for each edge $e \in E(G)$ as follows: Let $r \geq 1$ be any fixed constant. For each i, $1 \leq i \leq n$, we set $\mathsf{cost}((v_s, v_i)) = 0$, $\mathsf{cost}((v_i, v_t)) = w(a_i)$, $\mathsf{vul}((v_s, v_i)) = \mathsf{vul}((v_i, v_t)) = r$, and $\mathsf{cap}((v_s, v_i)) = \mathsf{cap}((v_i, v_t)) = r + p(a_i)$. Clearly, G is a bipartite series-parallel graph with $\mathsf{cap}(e) \geq 1$ and $\mathsf{vul}(e) = r$ for all edges $e \in E(G)$, as shown in Fig. 2. Finally, we set the number k of required $v_s v_t$-paths as $k = nr + d$, and the upper bound c on the penalty as the same upper bound on the total weight (i.e., capacity) of Knapsack. This corresponding instance can be constructed in polynomial time.

We show that a given instance of Knapsack is a yes-instance if and only if the corresponding instance of the minimum vulnerability problem is a yes-instance.

Suppose that a given instance of Knapsack is a yes-instance. Then, there exists a set $A' \subseteq A$ such that $\sum_{a_i \in A'} w(a_i) \leq c$ and $\sum_{a_i \in A'} p(a_i) \geq d$. In this case, a feasible solution of the minimum vulnerability problem can be obtained by the following steps. First, for each vertex v_i, $1 \leq i \leq n$, we choose the number r

of v_sv_t-paths that pass through two edges (v_s, v_i) and (v_i, v_t). Since the threshold for each edge in $E(G)$ is set to r, there is no penalty for these nr paths. Then, for each item $a_i \in A'$, we additionally choose the number $p(a_i)$ of v_sv_t-paths via the corresponding vertex v_i, and pay the penalty for each of the edges (v_s, v_i) and (v_i, v_t). Thus, the total penalty is $\sum_{a_i \in A'} \{\mathsf{cost}((v_s, v_i)) + \mathsf{cost}((v_i, v_t))\} = \sum_{a_i \in A'} w(a_i) \leq c$. Since $\sum_{a_i \in A'} p(a_i) \geq d$, the total number of chosen v_sv_t-paths is at least $nr + d = k$. Therefore, the chosen v_sv_t-paths form a feasible solution, and hence the corresponding instance of the minimum vulnerability problem is a yes-instance.

Conversely, suppose that the corresponding instance of the minimum vulnerability problem is a yes-instance. Then, there exists a set $\mathcal{P}$ consisting of $k = nr + d$ of v_sv_t-paths on G such that the penalty $\lambda(\mathcal{P})$ of $\mathcal{P}$ is at most c. Let $B \subseteq V(G)$ be the set of all vertices v_i in G such that the edges (v_s, v_i) and (v_i, v_t) incident to v_i are passed through by more than r paths in $\mathcal{P}$. Namely, we have to pay the penalties for the edges (v_s, v_i) and (v_i, v_t) if and only if $v_i \in B$. Let A' be the set of all items in A that correspond to the vertices in B. Then, we clearly have $\sum_{a_i \in A'} w(a_i) = \sum_{v_i \in B} \{\mathsf{cost}((v_s, v_i)) + \mathsf{cost}((v_i, v_t))\}$, and hence $\sum_{a_i \in A'} w(a_i) \leq c$. Since we pay the penalties for the edges (v_s, v_i) and (v_i, v_t) such that $v_i \in B$, the total number of v_sv_t-paths passing through these edges is more than $\sum_{v_i \in B} \mathsf{vul}((v_s, v_i))$ and at most $\sum_{v_i \in B} \mathsf{cap}((v_s, v_i))$. On the other hand, for the edges (v_s, v_i) and (v_i, v_t) such that $v_i \notin B$, we do not pay the penalties for them, and hence the total number of v_sv_t-paths passing through these edges is at most $\sum_{v_i \in V(G) \setminus B} \mathsf{vul}((v_s, v_i))$. Therefore, we have

$$\sum_{v_i \in B} \mathsf{cap}((v_s, v_i)) + \sum_{v_i \in V(G) \setminus B} \mathsf{vul}((v_s, v_i)) \geq |\mathcal{P}| = nr + d.$$

Since $\mathsf{cap}((v_s, v_i)) = r + p(a_i)$ and $\mathsf{vul}((v_s, v_i)) = r$,

$$\begin{aligned} \sum_{v_i \in B} \mathsf{cap}((v_s, v_i)) + \sum_{v_i \in V(G) \setminus B} \mathsf{vul}((v_s, v_i)) &= \sum_{a_i \in A'} p(a_i) + \sum_{a_i \in A} r \\ &= \sum_{a_i \in A'} p(a_i) + nr. \end{aligned}$$

Therefore, $\sum_{a_i \in A'} p(a_i) \geq d$ holds for the subset A' of items. Thus, the set A' is a feasible solution for the given instance of Knapsack, and hence it is a yes-instance. □

2.2 Threshold graphs

A graph G is a *threshold graph* if there exists a real number α and a mapping $w : V(G) \to \mathbb{R}$ such that $(x, y) \in E(G)$ if and only if $w(x) + w(y) \geq \alpha$, where $\mathbb{R}$ is the set of all real numbers [4].

Theorem 2. *The minimum vulnerability problem is NP-hard for threshold graphs, even if* $\mathsf{cap}(e) \geq 1$ *and* $\mathsf{vul}(e) = r$ *hold for all edges* $e \in E(G)$*, where* $r \geq 1$ *is any fixed constant.*

Proof. We modify the instance constructed in the proof of Theorem 1, as follows: Add an edge $e = (v_s, v_t)$ to the graph and set $\mathsf{cost}(e) = 1$, $\mathsf{vul}(e) = r$ and $\mathsf{cap}(e) = r$. (See Fig. 3.) Then, reset the number k of required $v_s v_t$-paths to $k = (n+1)r + d$. Clearly, the graph is a threshold graph such that $\mathsf{cap}(e) \geq 1$ and $\mathsf{vul}(e) = r \geq 1$ hold for all edges $e \in E(G)$.

Note that the edge (v_s, v_t) can be passed through by at most r paths, and these r paths do not cause any extra penalty. Therefore, the same arguments in the proof of Theorem 1 establish the theorem. □

3 Algorithm for Bounded Treewidth Graphs

In this section, we give an algorithm for bounded treewidth graphs.

A *tree-decomposition* of a graph G is a pair $\langle \{X_i : i \in V_T\}, T \rangle$, where $T = (V_T, E_T)$ is a rooted tree, such that the following four conditions (1)–(4) hold [2]:

(1) Each X_i is a subset of $V(G)$, and is called a *bag*;
(2) $\bigcup_{i \in V_T} X_i = V(G)$;
(3) for each edge $(u, v) \in E(G)$, there is at least one node $i \in V_T$ such that $u, v \in X_i$; and
(4) for each vertex $v \in V(G)$, the set $\{i \in V_T : v \in X_i\}$ induces a connected subgraph in T.

For example, Fig. 4(b) illustrates a tree-decomposition of the graph G in Fig. 4(a). We will refer to a *node* in V_T in order to distinguish it from a vertex in $V(G)$. The *width* of a tree-decomposition $\langle \{X_i : i \in V_T\}, T \rangle$ is defined as $\max\{|X_i| - 1 : i \in V_T\}$, and the *treewidth* of G is the minimum t such that G has a tree-decomposition of width t. We denote by $\mathsf{tw}(G)$ the treewidth of G.

Recall that the minimum vulnerability problem is NP-hard even for series-parallel graphs (Theorem 1), which are of treewidth at most two. In this section, we thus give a pseudo-polynomial-time algorithm for bounded treewidth graphs.

Theorem 3. *Let G be a graph whose treewidth is bounded by a fixed constant t. Then, $\mathsf{OPT}_k(G)$ can be computed in time $(k+1)^{O(t^{t+1})} n$, where $n = |V(G)|$.*

As a proof of Theorem 3, we give such an algorithm in the remainder of this section. Our algorithm can easily be modified to actually find k $v_s v_t$-paths on G with the minimum penalty $\mathsf{OPT}_k(G)$.

3.1 Nice tree-decomposition

A tree-decomposition $\langle \{X_i : i \in V_T\}, T \rangle$ of G is called a *nice tree-decomposition* if the following conditions (5)–(10) hold [2]:

(5) $|V_T| = O(t^2 n)$, where $n = |V(G)|$ and $t = \max\{|X_i| - 1 : i \in V_T\}$;
(6) every node in V_T has at most two children in T;
(7) if a node $i \in V_T$ has two children l and r, then $X_i = X_l = X_r$;
(8) if a node $i \in V_T$ has only one child j, then
   - $|X_i| = |X_j| - 1$ and $X_i \subset X_j$ (such a node i is called a *forget* node); or

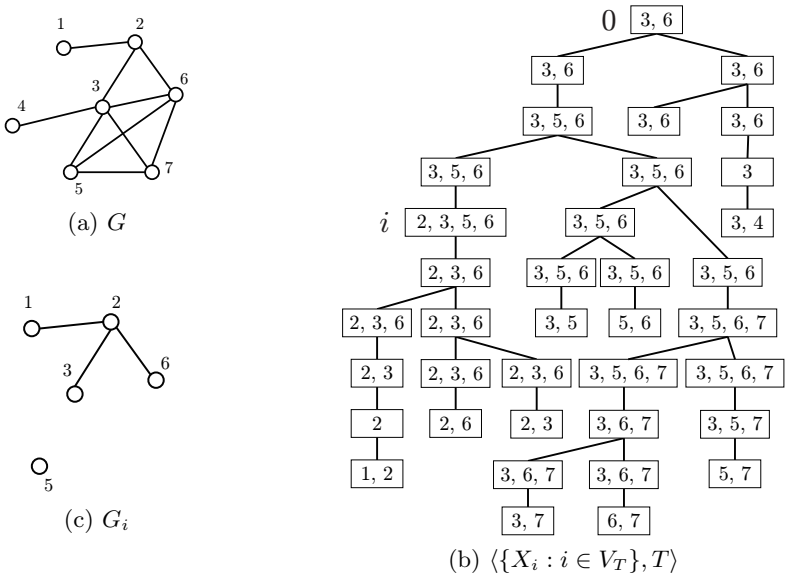

Fig. 4. (a) Graph G, (b) a nice tree-decomposition $\langle\{X_i : i \in V_T\}, T\rangle$ of G, and (c) the subgraph G_i of G for the node $i \in V_T$.

- $|X_i| = |X_j| + 1$ and $X_i \supset X_j$ (such a node i is called an *introduce* node);

(9) for each edge $(u, v) \in E(G)$, there is a leaf node $i \in V_T$ such that $u, v \in X_i$; and

(10) the bag of every leaf node in V_T contains exactly two vertices.

Figure 4(b) illustrates a nice tree-decomposition $\langle\{X_i : i \in V_T\}, T\rangle$ of the graph G in Fig. 4(a) whose treewidth is three. Let t be a fixed constant. Then, for a given graph G, there is a linear-time algorithm which either outputs $\mathsf{tw}(G) > t$ or gives a nice tree-decomposition of G whose width is at most t [2].

Given a graph G, we assume without loss of generality that $(v_s, v_t) \in E(G)$. If $(v_s, v_t) \notin E(G)$, then add (v_s, v_t) to $E(G)$ and set $\mathsf{cap}((v_s, v_t)) = 0$. The treewidth of this graph is bounded by $\mathsf{tw}(G) + 2$.

Let $\langle\{X_i : i \in V_T\}, T\rangle$ be a nice tree-decomposition with width at most t of a graph G. We regard the node i with $X_i = \{v_s, v_t\}$ as the root 0 of T. For each edge $e = (u, v) \in E(G)$, since there are some leaves whose bags contain both u and v, we arbitrarily choose one of such bags, say X_i, which is called a *representation* of e and denoted by $\mathsf{rep}(e) = i$.

We recursively define a vertex set $V_i \subseteq V(G)$ and an edge set $E_i \subseteq E(G)$ for each node i of T, similar to the way used in [7], as follows:

(a) If i is a leaf, then let $V_i = X_i$ and $E_i = \{e \in E(G) : \mathsf{rep}(e) = i\}$; and

(b) if i is an internal node having only one child j, then let $V_i = V_j \cup X_i$ and $E_i = E_j$, where V_j and E_j are the vertex and edge sets for j, respectively; and

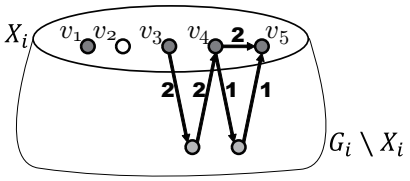

Fig. 5. An example of an $(\boldsymbol{s}, \boldsymbol{a})$-path set.

(c) if i is an internal node having two children l and r, then let $V_i = V_l \cup V_r$ and $E_i = E_l \cup E_r$, where V_l and E_l are the vertex and edge sets for l, respectively, and V_r and E_r are the vertex and edge sets for r, respectively.

Then, for each node i of T, we denote by G_i the graph with vertex set V_i and edge set E_i and hence $G_i = (V_i, E_i)$. In this way, for any node i with the children l and r, there exists no edge that is contained in both E_r and E_l.

3.2 Definitions

Let $\mathcal{S}(X_i)$ be the set of all permutations of the vertices in X_i, and let $\mathcal{A}(X_i)$ be the set of all $|X_i|$-tuples of nonnegative integers at most k. A path P in G_i joining two vertices $v, v' \in X_i$ is *inner* if $V(P) \setminus \{v, v'\} \subseteq V_i \setminus X_i$. For a pair of $\boldsymbol{s} = (v_1, v_2, \ldots, v_{|X_i|}) \in \mathcal{S}(X_i)$ and $\boldsymbol{a} = (a_1, a_2, \ldots, a_{|X_i|-1}) \in \mathcal{A}(X_i)$, a set $\mathcal{P}$ of paths in G_i is called an $(\boldsymbol{s}, \boldsymbol{a})$-*path set* if $|\mathcal{P}| = \sum_{j=1}^{|X_i|-1} a_j$ and $\mathcal{P}$ has exactly a_j inner $v_j v_{j+1}$-paths for each j, $1 \leq j \leq |X_i| - 1$. Figure 5 illustrates an $(\boldsymbol{s}, \boldsymbol{a})$-path set, where $\boldsymbol{s} = (v_1, v_3, v_4, v_5, v_2)$ and $\boldsymbol{a} = (0, 2, 3, 0)$. Let $\pi(X_i)$ be a set of pairs $(\boldsymbol{s}, \boldsymbol{a})$ such that $\boldsymbol{s} \in \mathcal{S}(X_i)$ and $\boldsymbol{a} \in \mathcal{A}(X_i)$. Then the set $\pi(X_i) = \{(\boldsymbol{s}_1, \boldsymbol{a}_1), (\boldsymbol{s}_2, \boldsymbol{a}_2), \ldots, (\boldsymbol{s}_{|\pi(X_i)|}, \boldsymbol{a}_{|\pi(X_i)|})\}$ is *active* if there exists a set $\mathcal{P}$ of inner paths in G_i and a partition $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{|\pi(X_i)|}$ of $\mathcal{P}$ such that each $\mathcal{P}_j$, $1 \leq j \leq |\pi(X_i)|$, forms an $(\boldsymbol{s}_j, \boldsymbol{a}_j)$-path set. Such a set $\mathcal{P}$ is called a $\pi(X_i)$-*path set*. Finally, for a set $\pi(X_i)$, we define

$$\lambda(\pi(X_i)) = \min\{\lambda(\mathcal{P}) : \mathcal{P} \text{ is a } \pi(X_i)\text{-path set}\}$$

if $\pi(X_i)$ is active and hence there exists a $\pi(X_i)$-path set, otherwise we let $\lambda(\pi(X_i)) = +\infty$.

Our algorithm computes $\lambda(\pi(X_i))$ for all sets $\pi(X_i)$ for each bag X_i of T, from the leaves of T to the root 0 of T, by means of dynamic programming. Then, $\mathsf{OPT}_k(G)$ can be computed at the root 0 from the values $\lambda(\pi(X_0))$, as described in Section 3.3.

3.3 Algorithm

In this subsection, we explain how to compute all values $\lambda(\pi(X_i))$ for each node $i \in V_T$ of T and all sets $\pi(X_i)$ for X_i. More specifically, we first compute all

values $\lambda(\pi(X_i))$ for each leaf i of T, and then compute $\lambda(\pi(X_i))$ for each internal node i in T. Finally, after computing all $\lambda(\pi(X_0))$ for the root 0 of T, we compute $\mathsf{OPT}_G(k)$.

[1. The node i is a leaf of T.]

In this case, by the definition (10) of a nice tree-decomposition, there are exactly two vertices v_1 and v_2 in X_i, and $e = (v_1, v_2) \in E(G)$. Then, a set $\pi(X_i) \in 2^{\mathcal{S}(X_i)\times\mathcal{A}(X_i)}$ is active if and only if the following two conditions hold:

(i) $|\pi(X_i)| = 1$; and

(ii) $a_1 \leq \mathsf{cap}(e)$ for the pair $(\boldsymbol{s}, \boldsymbol{a}) \in \pi(X_i)$ with $\boldsymbol{a} = (a_1)$.

For an active set $\pi(X_i)$, we let

$$\lambda(\pi(X_i)) = \begin{cases} 0 & \text{if } 0 \leq a_1 \leq \mathsf{vul}(e); \\ \mathsf{cost}(e) & \text{if } \mathsf{vul}(e) < a_1 \leq \mathsf{cap}(e). \end{cases}$$

For the other sets $\pi(X_i)$, we let $\lambda(\pi(X_i)) = +\infty$.

[2. The node i is an internal node.]

Since $\langle \{X_i : i \in V_T\}, T\rangle$ is a nice tree-decomposition of G and the node i is an internal node, either i has two children, is a forget node, or is an introduce node. Therefore we have the following three cases to consider.

Case 1: The node i has two children l and r.

In this case, each set of paths in G_i can be obtained by merging two sets of paths in G_l and G_r. Therefore, a set $\pi(X_i) \in 2^{\mathcal{S}(X_i)\times\mathcal{A}(X_i)}$ is active if and only if there exist two active sets $\pi(X_l) \in 2^{\mathcal{S}(X_l)\times\mathcal{A}(X_l)}$ and $\pi(X_r) \in 2^{\mathcal{S}(X_r)\times\mathcal{A}(X_r)}$ such that for each $(\boldsymbol{s}, \boldsymbol{a}) \in \pi(X_i)$, there exist $(\boldsymbol{s}_l, \boldsymbol{a}_l) \in \pi(X_l)$ and $(\boldsymbol{s}_r, \boldsymbol{a}_r) \in \pi(X_r)$ satisfying $\boldsymbol{s} = \boldsymbol{s}_l = \boldsymbol{s}_r$ and $\boldsymbol{a} = \boldsymbol{a}_l + \boldsymbol{a}_r$, where $\boldsymbol{a}_l + \boldsymbol{a}_r$ is defined as the addition of each element of $\boldsymbol{a}_l$ and $\boldsymbol{a}_r$. Then, $\lambda(\pi(X_i)) = \min\{\lambda(\pi(X_l)) + \lambda(\pi(X_r))\}$, where the minimum is taken over all pairs of such active $\pi(X_l)$ and $\pi(X_r)$.

Case 2: The node i is a forget node.

Let j be the child of the node i, and let v be the vertex such that $X_j \setminus X_i = \{v\}$. Then, a set $\pi(X_i) \in 2^{\mathcal{S}(X_i)\times\mathcal{A}(X_i)}$ is active if and only if there exists an active set $\pi(X_j) \in 2^{\mathcal{S}(X_j)\times\mathcal{A}(X_j)}$ such that for each $(\boldsymbol{s}, \boldsymbol{a}) \in \pi(X_i)$, there exists some pair $(\boldsymbol{s}', \boldsymbol{a}') \in \pi(X_j)$ with $\boldsymbol{s}' = (v'_1, v'_2, \ldots, v'_{|X_j|})$ and $\boldsymbol{a}' = (a'_1, a'_2, \ldots, a'_{|X_j|-1})$ satisfying the following two conditions:

(i) $\boldsymbol{s} = \boldsymbol{s}' \setminus v$, where $\boldsymbol{s}' \setminus v$ is the sequence obtained from $\boldsymbol{s}'$ by deleting v; and

(ii) $a'_{l-1} = a'_l$ and $\boldsymbol{a} = (a'_1, a'_2, \ldots, a'_{l-1}, a'_{l+1}, \ldots, a'_{|X_j|-1})$ for the index l such that $v'_l = v$ in $\boldsymbol{s}'$.

Then, $\lambda(\pi(X_i))$ is the minimum value of $\lambda(\pi(X_j))$, taken over all such active sets $\pi(X_j)$.

Case 3: The node i is an introduce node.

Let j be the child of the node i. In this case, since $|X_i \setminus X_j| = 1$, let v be the vertex in $X_i \setminus X_j$. Then, a set $\pi(X_i) \in 2^{\mathcal{S}(X_i)\times\mathcal{A}(X_i)}$ is active if and only if the following two conditions hold:

(i) $a_{l-1} = a_l = 0$ for each pair $(\boldsymbol{s}, \boldsymbol{a}) \in \pi(X_i)$ such that $\boldsymbol{s} = (v_1, v_2, \ldots, v_{|X_i|})$ with $v_l = v$ and $\boldsymbol{a} = (a_1, a_2, \ldots, a_{|X_i|-1})$; and

(ii) there exists a set $\pi(X_j) \in 2^{\mathcal{S}(X_j)\times\mathcal{A}(X_j)}$ which is active such that $\pi(X_j) = \{(\boldsymbol{s} \setminus v, \boldsymbol{a}') \mid (\boldsymbol{s}, \boldsymbol{a}) \in \pi(X_i)\}$, where $v_l = v$ in $\boldsymbol{s} = (v_1, v_2, \ldots, v_{|X_i|})$, $\boldsymbol{a} = (a_1, a_2, \ldots, a_{|X_i|-1})$ and $\boldsymbol{a}' = (a_1, a_2, \ldots, a_{l-1}, a_{l+1}, \ldots, a_{|X_i|-1})$.

Then, $\lambda(\pi(X_i))$ is the minimum value of $\lambda(\pi(X_j))$, taken over all such active sets $\pi(X_j)$.

[3. The node i is the root of T.]

In this case, $i = 0$. We first compute the values $\lambda(\pi(X_0))$ for all sets $\pi(X_0)$ for X_0, according to one of the three cases 2–4 above. Then, our algorithm computes $\mathsf{OPT}_k(G)$ from the values $\lambda(\pi(X_0))$ for all active sets $\pi(X_0) \in 2^{\mathcal{S}(X_0)\times\mathcal{A}(X_0)}$. Since $X_0 = \{v_s, v_t\}$, we only need to count the number of inner paths connecting v_s and v_t. Therefore, $\mathsf{OPT}_k(G) = \min \lambda(\pi(X_0))$, where the minimum is taken over all active sets $\pi(X_0) = \{(\boldsymbol{s}, \boldsymbol{a})\} \in 2^{\mathcal{S}(X_0)\times\mathcal{A}(X_0)}$ such that $\boldsymbol{a} = (k)$.

3.4 Running time

Recall that $|X_i| \leq t + 1$ for each node $i \in V_T$, where t is an upper bound on the treewidth of G. Then, $|\mathcal{S}(X_i)| = (t+1)! = O\big(t^{(t+1)}\big)$. Furthermore, since $\mathcal{A}(X_i)$ is the set of all $|X_i|$-tuples of nonnegative integers at most k, we have $|\mathcal{A}(X_i)| \leq (k+1)^{(t+1)}$. Thus, the number of all sets $\pi(X_i) \in 2^{\mathcal{S}(X_i)\times\mathcal{A}(X_i)}$ for each node $i \in V_T$ can be bounded by

$$\big((k+1)^{(t+1)}\big)^{(t+1)!} \leq (k+1)^{O\left(t^{(t+1)}\right)}.$$

Recall that $|X_i| = 2$ for each leaf $i \in V_T$. Then, according to the case 1 above, one can compute the value $\lambda(\pi(X_i))$ in $O(1)$ time for each set $\pi(X_i)$. Therefore, the values $\lambda(\pi(X_i))$ for all sets $\pi(X_i) \in 2^{\mathcal{S}(X_i)\times\mathcal{A}(X_i)}$ can be computed in $(k+1)^{O\left(t^{(t+1)}\right)}$ time. By the definition (5) of a nice tree-decomposition, T has at most $O(t^2 n)$ leaves; one can thus compute $\lambda(\pi(X_i))$ for all leaves of T in $n(k+1)^{O\left(t^{(t+1)}\right)}$ time.

Similarly, for each internal node i, each of the update formulas above can be computed in $(k+1)^{O\left(t^{(t+1)}\right)}$ time for each set $\pi(X_i)$. Therefore, the values $\lambda(\pi(X_i))$ for all sets $\pi(X_i) \in 2^{\mathcal{S}(X_i)\times\mathcal{A}(X_i)}$ can be computed in $(k+1)^{O\left(t^{(t+1)}\right)}$ time for each internal node i. By the definition (5) of a nice tree-decomposition, T has at most $O(t^2 n)$ internal nodes, and hence one can compute the values $\lambda(\pi(X_i))$ for all internal nodes of T in $n(k+1)^{O\left(t^{(t+1)}\right)}$ time.

Finally, for the root 0 of T, we can compute $\mathsf{OPT}_k(G)$ in $(k+1)^{O\left(t^{(t+1)}\right)}$ time from the values $\lambda(\pi(X_0))$.

In this way, our algorithm runs in $n(k+1)^{O\left(t^{(t+1)}\right)}$ time in total. This completes the proof of Theorem 3. □

4 Parameterized Algorithm for Chordal Graphs

A graph G is *chordal* if every cycle in G of length at least four has a chord, which is an edge joining non-consecutive vertices in the cycle [4].

Recall that the minimum vulnerability problem is NP-hard for threshold graphs, which form a subclass of chordal graphs, even when $\mathsf{vul}(e) \geq 1$ and $\mathsf{cap}(e) \geq 1$ hold for all edges $e \in E(G)$ (Theorem 2). In this section, we thus give an FPT algorithm for chordal graphs when parameterized by the number k of required $v_s v_t$-paths.

Theorem 4. *Let G be a chordal graph with n vertices and m edges such that $\mathsf{vul}(e) \geq 1$ and $\mathsf{cap}(e) \geq 1$ hold for all edges $e \in E(G)$. Then, $\mathsf{OPT}_k(G)$ can be computed in time $m + (k+1)^{O\left(k^{(3k+2)}\right)} n$.*

As a proof of Theorem 4, we give such an algorithm in the remainder of this section. If a graph G is a chordal graph, then there exists a tree-decomposition $\langle \{X_i : i \in V_T\}, T \rangle$ such that each bag X_i forms a clique, and such a tree-decomposition can be found in linear time [3].

For a vertex subset V' of a graph G, let $G[V']$ be the subgraph of G induced by V'. First, we prove the following two lemmas.

Lemma 1. *Let G be a graph with a cut-set X such that $\mathsf{vul}(e) \geq 1$ and $\mathsf{cap}(e) \geq 1$ hold for all edges $e \in E(G)$. Suppose that X is a clique and $|X| \geq 3k$. If there is a connected component C in $G \setminus X$ such that $v_s, v_t \in V(C) \cup X$, then $\mathsf{OPT}_k(G, v_s, v_t) = \mathsf{OPT}_k(G', v_s, v_t)$, where $G' = G[V(C) \cup X]$.*

Proof. Since G' is a subgraph of G, we have $\mathsf{OPT}_k(G, v_s, v_t) \leq \mathsf{OPT}_k(G', v_s, v_t)$. Therefore, it suffices to prove that $\mathsf{OPT}_k(G, v_s, v_t) \geq \mathsf{OPT}_k(G', v_s, v_t)$.

Let G_X be the graph obtained from G by contracting all vertices in X into one vertex v_X. Each resulting edge (v, v_X) has the same values of vulnerability, capasity and cost as its original edge. Note that if $v_s \in X$ then let $v_s = v_X$, and if $v_t \in X$ then let $v_t = v_X$. Clearly $\mathsf{OPT}_k(G, v_s, v_t) \geq \mathsf{OPT}_k(G_X, v_s, v_t)$, and hence it suffices to prove that $\mathsf{OPT}_k(G_X, v_s, v_t) \geq \mathsf{OPT}_k(G', v_s, v_t)$.

Let $\mathcal{P}_X$ be a set of k $v_s v_t$-paths as an optimal solution in G_X. Then we will construct a set $\mathcal{P}'$ of k $v_s v_t$-paths in G' from $\mathcal{P}_X$ such that $\lambda(\mathcal{P}') = \lambda(\mathcal{P}_X)$ as follows. For each path $P \in \mathcal{P}_X$, there are the following two cases to consider.

Case 1: The edges in G' corresponding to the edges on P in G_X form a $v_s v_t$-path P' in G'.

In this case we add the path P' to $\mathcal{P}'$.

Case 2: The edges in G' corresponding to the edges on P in G_X form exactly two paths in G'; one is a $v_s v_1$-path P_1 and the other is a $v_2 v_t$-path P_2, where $v_1, v_2 \in X$.

In this case, we choose an arbitrary vertex $u \in X$ which is not on any path in $\mathcal{P}'$ so far and is not an end of some edges corresponding to the edges on some paths in $\mathcal{P}_X$. Since $|\mathcal{P}_X| = k$, there are at most $2k$ edges on paths in $\mathcal{P}_X$ adjacent to vertices in X. Furthermore $|X| \geq 3k$. Therefore, there are at least

$|X| - 2k \geq k$ vertices in X which can be chosen in total. After chosen, by adding two edges $e_1 = (v_1, u)$ and $e_2 = (u, v_2)$ to join P_1 to P_2, the resulting path P' is a $v_s v_t$-path in G', and add it to $\mathcal{P}'$. Since $\mathsf{vul}(e_1) \geq 1$, $\mathsf{cap}(e_1) \geq 1$, $\mathsf{vul}(e_2) \geq 1$, $\mathsf{cap}(e_2) \geq 1$, and these two edges e_1 and e_2 are not used by another path in $\mathcal{P}'$, we have $\lambda(\mathcal{P}') = \lambda(\mathcal{P}_X)$.

We have completed to construct a set $\mathcal{P}'$ of k $v_s v_t$-paths in G' from $\mathcal{P}_X$ such that $\lambda(\mathcal{P}') = \lambda(\mathcal{P}_X)$, and hence $\mathsf{OPT}_k(G', v_s, v_t) \leq \lambda(\mathcal{P}') = \lambda(\mathcal{P}_X) = \mathsf{OPT}_k(G_X, v_s, v_t)$. □

Lemma 2. *Let G be a graph with a cut-set X such that $\mathsf{vul}(e) \geq 1$ and $\mathsf{cap}(e) \geq 1$ hold for all edges $e \in E(G)$. Suppose that X is a clique and $|X| \geq 3k$. If there are two connected components C_1 and C_2 in $G \setminus X$, then $\mathsf{OPT}_k(G, v_s, v_t) = \mathsf{OPT}_k(G_1, v_s, v) + \mathsf{OPT}_k(G_2, v, v_t)$, where v is an arbitrary vertex in $X \setminus \{v_s, v_t\}$, $G_1 = G[V(C_1) \cup X]$ and $G_2 = G[V(C_2) \cup X]$.*

Proof. Since G_1 and G_2 are subgraphs of G, it is trivial that $\mathsf{OPT}_k(G, v_s, v_t) \leq \mathsf{OPT}_k(G_1, v_s, v) + \mathsf{OPT}_k(G_2, v, v_t)$. Therefore, it suffices to prove that

$$\mathsf{OPT}_k(G, v_s, v_t) \geq \mathsf{OPT}_k(G_1, v_s, v) + \mathsf{OPT}_k(G_2, v, v_t).$$

Let G_X be the graph obtained from G by contracting all vertices in X as one vertex v_X. Each resulting edge (v, v_X), $v \in V(G_X) \setminus \{v_X\}$, has the same values of vulnerability, capasity and cost of its corresponding edge. Note that if $v_s \in X$ then let $v_s = v_X$, and if $v_t \in X$ then let $v_t = v_X$. Clearly $\mathsf{OPT}_k(G, v_s, v_t) \geq \mathsf{OPT}_k(G_X, v_s, v_t)$, and hence it suffices to prove that

$$\mathsf{OPT}_k(G_X, v_s, v_t) \geq \mathsf{OPT}_k(G_1, v_s, v) + \mathsf{OPT}_k(G_2, v, v_t).$$

Let $\mathcal{P}_X$ be a set of k $v_s v_t$-paths as an optimal solution in G_X. Then we will construct a set $\mathcal{P}_1$ of k $v_s v$-paths in G_1, and a set $\mathcal{P}_2$ of k $v v_t$-paths in G_2, such that $\lambda(\mathcal{P}_X) = \lambda(\mathcal{P}_1) + \lambda(\mathcal{P}_2)$ as follows.

For each path $P_X \in \mathcal{P}_X$, the edges in $E(G_1) \cup E(G_2)$ corresponding to the edges on P_X in G_X form exactly two paths: One is a $v_s v_1$-path P_1 in G_1, and the other is a $v_2 v_t$-path P_2 in G_2, where $v_1, v_2 \in X$.

Then, we choose an arbitrary vertex $u \in X$ which is not on any path in $\mathcal{P}_1$ so far and is not an end of some edges corresponding to the edges on some paths in $\mathcal{P}_X$. Since $|\mathcal{P}_X| = k$, there are at most $2k$ edges on paths in $\mathcal{P}_X$ adjacent to vertices in X. Furthermore $|X| \geq 3k$. Therefore there are at least $|X| - 2k \geq k$ vertices in X which can be chosen in total. After chosen, by adding two edges $e_1 = (v_1, u)$ and $e_2 = (u, v)$ to join P_1 to v, the resulting path P_1 is a $v_s v$-path in G_1, and add it to $\mathcal{P}_1$. Since $\mathsf{vul}(e_1) \geq 1$, $\mathsf{cap}(e_1) \geq 1$, $\mathsf{vul}(e_2) \geq 1$, $\mathsf{cap}(e_2) \geq 1$, and these two edges e_1 and e_2 are not used by another path in $\mathcal{P}_1$, we do not pay any costs on e_1 and e_2.

Similarly, we choose an arbitrary vertex $u \in X$ which is not on any path in $\mathcal{P}_2$ so far and is not an end of some edges corresponding to the edges on some paths in $\mathcal{P}_X$. After chosen, by adding two edges $e_1 = (v, u)$ and $e_2 = (u, v_2)$ to join P_2 to v, the resulting path P_2 is a $v v_t$-path in G_2, and add it to $\mathcal{P}_2$. Since

these two edges e_1 and e_2 are not used by another path in $\mathcal{P}_2$, we do not pay any costs on e_1 and e_2.

We have completed to construct two sets, $\mathcal{P}_1$ of k $v_s v$-paths in G_1 and $\mathcal{P}_2$ of k vv_t-paths in G_2 from $\mathcal{P}_X$ such that $\lambda(\mathcal{P}_1) + \lambda(\mathcal{P}_2) = \lambda(\mathcal{P}_X)$, and hence

$$\begin{aligned}\mathsf{OPT}_k(G_1, v_s, v) + \mathsf{OPT}_k(G_2, v, v_t) &\leq \lambda(\mathcal{P}_1) + \lambda(\mathcal{P}_2) \\ &= \lambda(\mathcal{P}_X) \\ &= \mathsf{OPT}_k(G, v_s, v_t).\end{aligned}$$

This completes the proof of Lemma 2. □

By using Lemmas 1 and 2, we thus have the following FPT algorithm $\mathsf{Alg}(G, v_s, v_t, k)$ that returns $\mathsf{OPT}_k(G, v_s, v_t)$ for a chordal graph G.

Algorithm 1 $\mathsf{Alg}(G, v_s, v_t, k)$

1: let $\langle \{X_i : i \in V_T\}, T \rangle$ be a tree-decomposition of the chordal graph G with treewidth t.
2: **if** there are all $i \in V_T$ such that $|X_i| \leq 3k - 1$ **then**
3: compute $\mathsf{OPT}_k(G, v_s, v_t)$ by Theorem 3 and return it;
4: **else**
5: /* in this case, there is a node $i \in V_T$ such that $|X_i| \geq 3k$ */
6: Let i be a node in V_T such that $|X_i| \geq 3k$;
7: **if** $v_s, v_t \in V(G_i)$ **then**
8: **return** $\mathsf{Alg}(G_i, v_s, v_t, k)$;
9: **else if** $v_s, v_t \notin V(G_i)$ **then**
10: **return** $\mathsf{Alg}(\overline{G_i}, v_s, v_t, k)$, where $\overline{G_i} = G[(V(G) \setminus V(G_i)) \cup X_i]$;
11: **else**
12: suppose without loss of generality that $v_s \in V(G_i)$ and $v_t \notin V(G_i)$;
13: let v be an arbitrary vertex in $X_i \setminus \{v_s, v_t\}$;
14: **return** $\mathsf{Alg}(G_i, v_s, v, k) + \mathsf{Alg}(\overline{G_i}, v, v_t, k)$;
15: **end if**
16: **end if**

We are now ready to prove Theorem 4.

Proof of Theorem 4. By Lemmas 1 and 2 and Theorem 3, $\mathsf{Alg}(G, v_s, v_t, k)$ above correctly computes $\mathsf{OPT}_k(G)$. Therefore, it suffices to prove the time-complexities. Lines 4–15 can be performed in combined $O(tn)$ time. By Theorem 3, Line 2–3 can be done in $(k+1)^{O(k^{(3k+2)})}n$ time. This completes the proof of Theorem 4. □

References

1. Assadi, S., Emamjomeh-Zadeh, E., Norouzi-Fard, A., Yazdanbod, S., Zarrabi-Zadeh, H.: The minimum vulnerability problem. In: Proc. ISAAC 2012, LNCS 7676, pp. 382–391 (2012)
2. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Computing 25, pp. 1305–1317 (1996)
3. Boutiche, M.A., Ait Haddadène, H., Le Thi, H.A.: Maintaining graph properties of weakly chordal graphs. Applied Mathematical Sciences 6, pp. 765–778 (2012)
4. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey, SIAM (1999)
5. Even, G., Kortsarz, G., Slany, W.: On network design problems: fixed cost flows and the covering steiner problem. ACM Trans. Algorithms 1, pp. 74–101 (2005)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco (1979)
7. Isobe, S., Zhou, X., Nishizeki, T.: A polynomial-time algorithm for finding total colorings of partial k-trees. International J. Foundations of Computer Science 10, pp. 171–194 (1999)
8. Krumke, S.O., Noltemeier, H., Schwarz, S., Wirth, H.-C., Ravi, R.: Flow improvement and network flows with fixed costs. In: Proc. OR 1998, pp. 158–167 (1998)
9. Omran, M.T., Sack, J.-R., Zarrabi-Zadeh, H.: Finding paths with minimum shared edges. J. Combinatorial Optimization 26, pp. 709–722 (2011)
10. Ye, Z.Q., Li, Y.M., Lu, H.Q., Zhou, X.: Finding paths with minimum shared edges in graphs with bounded treewidth. In: Proc. FCS 2013, pp. 40–46 (2013)