FPT Approximation using Treewidth: Capacitated Vertex Cover, Target Set Selection and Vector Dominating Set

Huairui Chu ✉
Nanjing University

Bingkai Lin ✉
Nanjing University

Abstract

Treewidth is a useful tool in designing graph algorithms. Although many NP-hard graph problems can be solved in linear time when the input graphs have small treewidth, there are problems which remain hard on graphs of bounded treewidth. In this paper, we consider three vertex selection problems that are W[1]-hard when parameterized by the treewidth of the input graph, namely the capacitated vertex cover problem, the target set selection problem and the vector dominating set problem. We provide two new methods to obtain FPT approximation algorithms for these problems. For the capacitated vertex cover problem and the vector dominating set problem, we obtain $(1 + o(1))$-approximation FPT algorithms. For the target set selection problem, we give an FPT algorithm providing a tradeoff between its running time and the approximation ratio.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases FPT approximation algorithm, Treewidth, Capacitated Vertex Cover, Target Set Selection, Vector Dominating Set

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

We consider problems whose goals are to select a minimum sized vertex set in the input graph that can "cover" all the target objects. In the capacitated vertex cover problem (CVC), we are given a graph G with a capacity function $c : V(G) \to \mathbb{N}$, the goal is to find a set $S \subseteq V(G)$ of minimum size such that every edge of G is covered[1] by some vertex in S and each vertex $v \in S$ covers at most $c(v)$ edges. This problem has application in planning experiments on redesign of known drugs involving glycoproteins [22]. In the target set selection problem (TSS), we are given a graph G with a threshold function $t : V(G) \to \mathbb{N}$. The goal is to select a minimum sized set $S \subseteq V(G)$ of vertices that can activate all the vertices of G. The activation process is defined as follows. Initially, all vertices in the selected set S are activated. In each round, a vertex v gets active if there are $t(v)$ activated vertices in its neighbors. The study of TSS has application in maximizing influence in social network [24]. Vector dominating set (VDS) can be regarded as a "one-round-spread" version of TSS, where the input consists of a graph G and a threshold function $t : V(G) \to \mathbb{N}$, and the goal is to find a set $S \subseteq V(G)$ such that for all vertices $v \in V$, there are at least $t(v)$ neighbors of v in S.

Since CVC generalizes the vertex cover problem, while TSS and VDS are no easier than the dominating set problem[2], they are both NP-hard and thus have no polynomial time

[1] An edge e can be covered by a vertex v if v is an endpoint of e.
[2] It is obvious that when $t(v) = 1$ for every vertex v in the graph, VDS is the dominating set problem. The reduction from dominating set to TSS can be found in [7].

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

algorithm unless $P = NP$. Polynomial time approximation algorithms for capacitated vertex cover problem have been studied extensively [22, 10, 20, 31, 9, 33, 32]. The problem has a 2-approximation polynomial time algorithm [20]. Assuming the *Unique Game Conjecture*, there is no polynomial time algorithm for the vertex cover problem with approximation ratio better than 2 [25]. As for the TSS problem, it is proved that the minimum version of TSS cannot be approximated to $2^{\log^{1-\epsilon} n}$ assuming $NP \nsubseteq DTIME(n^{polylog(n)})$ [8]. In [11] it is proved that VDS cannot be approximated within a factor of $c \ln n$ for some c unless $P = NP$.

Another way of dealing with hard computational problems is to use parameterized algorithms. For any input instance x with parameter k, an algorithm with running time upper bounded by $f(k) \cdot |x|^{O(1)}$ for some computable function f is called FPT. A natural parameter for a computational problem is the solution size. The first FPT algorithm with running time $1.2^{k^2} + n^2$ for capacitated vertex cover problem parameterized by solution size was provided in [23]. In [15], the authors gave an improved FPT algorithm with $k^{3k} \cdot |G|^{O(1)}$ running time. However, using the solution size as parameter might be too strict for CVC. Note that CVC instances with sublinear capacity functions cannot have small sized solutions. On the other hand, TSS parameterized by its solution size is W[P]-hard [3] according to [1]. VDS is W[2]-hard since it generalizes the dominating set problem.

In this paper, we consider these problems parameterized by the treewidth [30] of the input graph. In fact, since the treewidth of a graph having k-sized vertex cover is also upper-bounded by k [15], CVC parameterized by treewidth can be regarded as a natural generalization of CVC parameterized by solution size. And it is already proved in [15] that CVC parameterized only by the treewidth of its input graph has no FPT algorithm assuming $W[1] \neq FPT$. As for the TSS problem, it can be solved in $n^{O(w)}$ time for graphs with n vertices and treewidth bounded by w and has no $n^{o(\sqrt{w})}$-time algorithm unless ETH fails [3]. VDS is also $W[1]$-hard when parameterized by treewidth [4], however, it admits an FPT algorithm with respect to the combined parameter $(w + k)$[29].

Recently, the approach of combining parameterized algorithms and approximation algorithms has received increased attention [17]. It is natural to ask whether there exist FPT algorithms for these problems with approximation ratios better than that of the polynomial time algorithms. Lampis [26] proposed a general framework for approximation algorithms on tree decomposition. Using his framework, one can obtain algorithms for CVC and VDS which outputs a solution of size at most $opt(I)$ on input instance I but may slightly violate the capacity or the threshold requirement within a factor of $(1 \pm \epsilon)$. However, the framework of Lampis can not be directly used to find an approximation solution for these problems satisfying all the capacity or threshold requirement. The situation becomes worse in the TSS problem, as the error might propagate during the activation process. We overcome these difficulties and give positive answer to the aforementioned question. For the CVC and VDS problems, we obtain $(1 + o(1))$-approximation FPT algorithms respectively.

▶ **Theorem 1.** *There exists an algorithm, which takes a CVC instance $I = (G, c)$ and a tree decomposition $(T, \mathcal{X})$ with width w for G as input and outputs an integer $\hat{k}_{\min} \in [opt(I), (1 + O(1/(w^2 \log n)))opt(I)]$ in $O((w \log n)^{O(w)} n^{O(1)})$ time.*

▶ **Theorem 2.** *There exists an algorithm running in time $2^{O(w^5 \log w)} n^{O(1)}$ which takes input an instance $I = (G, t)$ of VDS and a tree decomposition of G with width w, finds a solution of size at most $(1 + O(1/w)) \cdot opt(I)$.*

[3] The well known W-hierarchy is $FPT \subseteq W[1] \subseteq W[2] \subseteq ... \subseteq W[P]$, where FPT denotes the set of problems who admits FPT algorithms. The basic conjecture on parameterized complexity is $FPT \neq W[1]$. We refer the readers to [16, 18, 13] for more details.

For the TSS problem, we give an approximation algorithm with a tradeoff between the approximation ratio and its running time.

▶ **Theorem 3.** *There is an algorithm which on input an instance $I = (G, t)$ of TSS and a tree decomposition of G with width w, finds a solution of size $(1 + (w + 1)/(C + 1)) \cdot opt(I)$ in time $n^{C+O(1)}$.*

Open problems and future work. Note that our FPT approximation algorithm for TSS has ratio equal to the treewidth of the input graph. An immediate question is whether this problem has $(1 + o(1)$-ratio parameterized approximation algorithm. We remark that the reduction from k-Clique to TSS in [3] does not preserve the gap. Thus it does not rule out constant FPT approximation algorithm for TSS on bounded treewidth graphs even under hypotheses like *parameterized inapproximablity hypothesis* (PIH) [27] or GAP-ETH [14, 28].

In the regime of exact algorithms, we have the famous Courcelle's Theorem which states that all problems defined in *monadic second order logic* have linear time algorithm on graphs of bounded treewidth [2, 12]. It is interesting to ask if one can obtain a similar algorithmic meta-theorem [21] for approximation algorithms.

1.1 Overview of our techiniques

Capacitated Vertex Cover. Our starting point is the exact algorithm for CVC on graphs with treewidth w in $n^{\Theta(w)}$ time. The exact algorithm has running time $n^{\Theta(w)}$ because it has to maintain a set of $(w + 1)$-dimension vectors $d : X_\alpha \to [n]$ for every node α in the tree decomposition. One can get more insight by checking out the exact algorithm for CVC in Section 3. To reduce the size of such a table, Lampis's approach [26] is to pick a parameter $\epsilon \in (0, 1)$ and round every integer to the closest integer power of $(1 + \epsilon)$. In other words, an integer n is represented by $(1 + \epsilon)^x$ with $(1 + \epsilon)^x \le n < (1 + \epsilon)^{x+1}$. Thus it suffices to keep $(\log n)^{O(w)}$ records for every bag in the tree decomposition. The price of this approach is that we can only have approximate values for records in the table. Note that the errors of approximate values might accumulate after addition (See Lemma 9). Nevertheless, we can choose a tree decomposition with height $O(w^2 \log n)$ and set $\epsilon = 1/poly(w \log n)$ so that if the dynamic programming procedure only involves adding and passing values of these vectors, then we can have $(1 + o(1))$-approximation values for all the records in the table.

Unfortunately, in the node of forgetting a vertex v, we need to compare the value of $d(v)$ and the capacity value $c(v)$. This task seems impossible if we do not have the exact value of $d(v)$. Our idea is to modify the "slightly-violating-capacity" solution, based on two crucial observations. The first is that, in a solution, for any vertex $v \in V$, the number of edges incident to v which are **not** covered by v presents a lower bound for the solution size. The second observation is that one can test if a "slightly-violating-capacity" solution can be turned into a good one in polynomial time. These observations are formally presented in Lemma 10 and 11.

Target Set Selection and Vector Dominating Set. We observe that both of the TSS and VDS problems are *monotone* and *splittable*, where the monotone property states that any super set of a solution is still a solution and the splittable property means that for any separator X of the input graph G, the union of X and solutions for components after removing X is also a solution for the graph G. We give a general approximation for vertex subset problems that are monotone and splittable. The key of our approximation algorithm is an observation that any bag in a tree decomposition is a separator in G. As the problem is splittable, we can design a procedure to find a bag, and remove it, which leads to a separation of G and we then deal with the component "rooted" by this bag. We can use this procedure repeatedly until the whole graph is done.

1.2 Organization of the Paper

In Section 2 the basic notations are given, and we formally define the problem we study. In Section 3 we present the exact algorithm for CVC. In Section 4 we present the approximate algorithm for CVC. In Section 5, we give the approximation algorithms for TSS and VDS.

2 Preliminaries

2.1 Basic Notations

We denote an undirected simple graph by $G = (V, E)$, where $V = [n]$ for some $n \in \mathbb{N}$ and $E \subseteq \binom{V}{2}$. Let $V(G) = V$ and $E(G) = E$ be its vertex set and edge set. For any vertex subset $S \subseteq V$, let the induced subgraph of S be $G[S]$. The edges of $G[S]$ are $E[S] = E(G) \cap \binom{S}{2}$. For any $S_1, S_2 \subseteq V$, we use $E[S_1, S_2]$ to denote the edge set between S_1 and S_2, i.e. $E[S_1, S_2] = \{e = (u, v) \in E | u \in S_1, v \in S_2\}$. For every $v \in V(G)$, we use $N(v)$ to denote the neighbors of v, and $d(v) := |N(v)|$.

For an orientation O of a graph G, which can be regarded as a directed graph whose underlying undirected graph is G, we use $D^+_O(v)$ to denote the outdegree of v and $D^-_O(v)$ its indegree. In a directed graph or an orientation, an edge (u, v) is said to start at u and sink at v. Reversing an edge is an operation, in which an edge (u, v) is replaced by (v, u).

In a graph $G = (V, E)$, a separator is a vertex set X such that $G[V \setminus X]$ is not a connected graph. In this case we say X separates V into disconnected parts $C_1, C_2, ... \subseteq V \setminus X$, where C_i and C_j are disconnected for all $i \neq j$ in $G[V \setminus X]$.

Let $f : A \to B$ be a mapping. For a subset $A' \subseteq A$, let $f[A']$ denote the mapping with domain A' and $f[A'](a) = f(a)$, for all $a \in A'$. Let $f \setminus a$ be $f[A \setminus \{a\}]$. For all $b \in B$, let $f^{-1}(b)$ be the set $\{a \in A' | f(a) = b\}$.

Let $\gamma \geq 0$ be a small value, we use $\mathbb{N}_\gamma$ to denote $\{0\} \cup \{(1 + \gamma)^x | x \in \mathbb{N}\}$. For $a, b \in \mathbb{R}$, we use $a \sim_\gamma b$ to denote that $b/(1 + \gamma) \leq a \leq (1 + \gamma)b$. It's easy to see this is a symmetric relation. Further, we use $[a]_\gamma$ to denote $\max_{x \in \mathbb{N}_\gamma, x \leq a} x$. Notice that $[a]_\gamma \sim_\gamma a$.

2.2 Problems

Capacitated Vertex Cover: An instance of CVC consists of a graph $G = (V, E)$ and a capacity function $c : V \to \mathbb{N}$ on the vertices. A solution is a pair (S, M) where $S \subseteq V$ and $M : E \to S$ is a mapping. If for all $v \in S, |M^{-1}(v)| \leq c(v)$ and for all $e \in E, M(e) \in e$, then we say that S is feasible. The size of a feasible solution is $|S|$. The goal of CVC is to find a feasible solution of minimum size. An equivalent description of this problem is the following. Let O be an orientation of all the edges in E. O is a feasible solution if and only if for all $v \in V, D^-_O(v) \leq c(v)$. The size of O is defined as $|\{v \in V | d^-(v) > 0\}|$. Here we actually use a directed edge (u, v) to represent that $\{u, v\}$ is covered by v. We mainly use this definition as it's more convenient for organizing our proof and analysis.

Target Set Selection: Given a graph $G = (V, E)$, a threshold function $t : V \to \mathbb{N}$, and a set $S \subseteq V$, the set $S' \subseteq V$ which contains the vertices activated by S is the set that:

- S' is the smallest set satisfying the following;
- $S \subseteq S'$;
- For a vertex v, if $|N(V) \cap S'| \geq t(v)$, then $v \in S'$.

One can find the vertices activated by S in polynomial time. Just start from $S' := S$, as long as there exists a vertex v such that $|N(v) \cap S'| \geq t(v)$, add v to S', until no such vertex exists. A vertex set that can activate all vertices in V is called a target set. The goal of TSS is to find a target set of minimum size.

Vector Dominating Set: Given a graph $G = (V, E)$, a threshold function $t : V \to \mathbb{N}$, the goal of Vector Dominating Set problem is to find a minimum vertex subset $S \subseteq V$ such that every vertex $v \in V \setminus S$ satisfies $|N(v) \cap S| \geq t(v)$.

2.3 Tree Decomposition

In this paper, we consider problems parameterized by the treewidth of the input graphs. A tree decomposition of a graph G is a pair $(T, \mathcal{X})$ such that

- T is a rooted tree and $\mathcal{X} = \{X_\alpha : \alpha \in V(T), X_\alpha \subseteq V(G)\}$ is a collection of subsets of $V(G)$;
- $\bigcup_{X_\alpha \in \mathcal{X}} X_\alpha = V(G)$;
- For every edge e of G, there exists an $X_\alpha \in \mathcal{X}$ such that $e \subseteq X_\alpha$;
- For every vertex v of G, the set $\{\alpha \in V(T) | v \in X_\alpha\}$ forms a subtree of T.

The width of a tree decomposition $(T, \mathcal{X})$ is $\max_{\alpha \in V(T)} |X_\alpha| - 1$. The treewidth of a graph G is the minimum width over all its tree decompositions.

It is convenient to work on a *nice tree decomposition*. Every node $\alpha \in V(T)$ in this nice tree decomposition is expected to be one of the following:

(i) Leaf Node: α is a leaf and $X_\alpha = \emptyset$;
(ii) Introducing v Node: α has exactly one child α_1, $v \notin X_{\alpha_1}$ and $X_\alpha = X_{\alpha_1} \cup \{v\}$;
(iii) Forgetting v Node: α has exactly one child α_1, $v \notin X_\alpha$ and $X_\alpha \cup \{v\} = X_{\alpha_1}$;
(iv) Join Node: α has exactly two children α_1, α_2 and $X_\alpha = X_{\alpha_1} = X_{\alpha_2}$.

We refer the reader to [13, 5] for more details of treewidth and nice tree decomposition. Using the tree balancing technique [6] and the method of introducing new nodes, we can transform any tree decomposition with width w in polynomial time into a nice tree decomposition with width $O(w)$, depth upper bounded by $O(w^2 \log n)$, and containing at most $O(nw)$ nodes. Moreover, we can add $O(w)$ nodes so that the root α_0 is assigned with an empty set $X_{\alpha_0} = \emptyset$. We assume all the nice tree decompositions discussed in this paper satisfy these properties.

The sets in $\mathcal{X}$ are called "bags". For a node $\alpha \in V(T)$, let T_α denote the subtree of T rooted by α. Let $V_\alpha \subseteq V$ denote the vertex set $V_\alpha = \cup_{\alpha' \in V(T_\alpha)} X_{\alpha'}$. Let $Y_\alpha := V_\alpha \setminus X_\alpha$. Notice that $X_{\alpha_0} = \emptyset$, so $Y_{\alpha_0} = V_{\alpha_0} = V$. For a node α, we use $\alpha_1(, \alpha_2)$ to denote its possible children. By the definition of tree decompositions, for a join node α, $Y_{\alpha_1} \cap Y_{\alpha_2} = \emptyset$.

3 Exact Algorithm for CVC

We present the exact algorithm for two reasons. The first is that one can gain some basic insights on the structure of the approximate algorithm by understanding the exact algorithm, which is more comprehensible. The other is that we need to compare the intermediate results of the exact algorithm and the approximate algorithm, so the total description of the algorithm can also be regarded as a recursive definition of the intermediate results (which are the sets R_α's defined in the following).

3.1 Definition of the Tables

Given a tree decomposition $(T, \mathcal{X})$, we run a classical bottom-up dynamic program to solve CVC. That is on each node α we allocate a record set R_α. R_α contains records of the form

(d,k). A record (d,k) consists of two elements: a mapping $d : X_\alpha \to \mathbb{N}$ and an integer $k \in \mathbb{N}$. At first, we present a definition of R_α by its properties. Then we define R_α according to the **Recursive Rules**. In Theorem 5 we in fact claim that these two definitions coincide.

Let G_α denote the graph with vertex set V_α and edge set $E[V_\alpha] \setminus E[X_\alpha]$. We expect that the table R_α has the following properties.

Expected Properties for R_α

A record $(d,k) \in R_\alpha$ if and only if there exists O, an orientation of G_α, such that

(1) For each $v \in X_\alpha$, $d(v) = D^+_O(v)$ is just its out degree;

(2) $D^-_O(v) \le c(v)$ for all $v \in Y_\alpha$;

(3) $|\{v \in Y_\alpha | D^-_O(v) > 0\}| \le k \le |Y_\alpha|$.

Intuitively, $(d,k) \in R_\alpha$ if there exists a vertex set $S \subseteq Y_\alpha$ and a mapping $M : E[V_\alpha] \setminus E[X_\alpha] \to S \cup X_\alpha$ such that

- all edges are covered correctly, i.e. $M(e) \in e$ for all $e \in E[V_\alpha] \setminus E[X_\alpha]$;
- for each $v \in X_\alpha$, there are $d(v)$ edges from v to Y_α that are covered by S, i.e. $|E[\{v\}, Y_\alpha] \cap \cup_{u \in S} M^{-1}(u)| = d(v)$;
- M satisfies the capacity constraints for vertices in Y_α, i.e. for all $v \in Y_\alpha$, $|M^{-1}(v)| \le c(v)$;
- $|S| \le k \le |Y_\alpha|$.

One can imagine that S is a feasible solution for a spanning subgraph of G_α, where the vector d governs the edges between X_α and Y_α.

Note that the root node α_0 satisfies $X_{\alpha_0} = \emptyset$, and $G_{\alpha_0} = G$. So if R_{α_0} is correctly computed, then the k values in those records in R_{α_0} have a one-to-one correspondence to all feasible solution sizes for the original instance. We output $\min_{(d,k) \in R_{\alpha_0}} k$ to solve the instance.

Recursive Rules for R_α

Fix a node $\alpha \in V(T)$, if α is a introducing node or a forgetting node, let α_1 be its child. If α is a join node, let α_1, α_2 be its children. In case α is a:

Leaf Node. $R_\alpha = \{(d,k)\}$, in which d is a mapping with empty domain and $k := 0$.

Introducing v Node. Note that by the properties of tree decompositions, there is no edge between v and Y_α in G. A record $(d,k) \in R_\alpha$ if and only if $(d \setminus v, k) \in R_{\alpha_1}$ and $d(v) = 0$.

Join Node. $(d,k) \in R_\alpha$ if and only if there exist $(d_1,k_1) \in R_{\alpha_1}$ and $(d_2,k_2) \in R_{\alpha_2}$ such that for all $v \in X_\alpha$, $d(v) = d_1(v) + d_2(v)$ and $k = k_1 + k_2$.

Forgetting v Node. $(d,k) \in R_\alpha$ if and only if there exists $(d_1,k_1) \in R_{\alpha_1}$ satisfying one of the following conditions:

(1) $k_1 = k$, $d_1(v) = |N(v) \cap Y_\alpha|$ and $d_1 \setminus v = d$. In this case, v is not "included in S". All the edges between v and Y_α must be covered by other vertices in Y_α.

(2) $k_1 = k-1$ and there exist $\Delta(v) \subseteq N(v) \cap X_\alpha$ and $A \in [|N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|, |N(v) \cap Y_\alpha|]$ such that $d_1(v) = A$, $d_1(u) = d(u) - 1$ for all $u \in \Delta(v)$, and $d_1(u) = d(u)$ for all $u \in X_{\alpha_1} \setminus (\Delta(v) \cup \{v\})$. In this case, v is "included in S". We enumerate a set $\Delta(v) \subseteq N(v) \cap X_\alpha$ of edges between v and X_α and let v cover these edges. Note that for a record $(d_1,k_1) \in R_{\alpha_1}$, there are $|N(v) \cap Y_\alpha| - d_1(v)$ edges that are covered by v. To construct (d,k) from (d_1,k_1), we need to check that $c(v) \ge |\Delta(v)| + |N(v) \cap Y_\alpha| - d_1(v)$, which is implicitly done by the setting $d_1(v) = A \ge |N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|$.

▶ Remark 4. In fact, one can find many different ways to define the dynamic programming table for CVC. We use this definition because we want to upper bound the values of records in R_α by the size of solution (Lemma 10), so we need to record "outdegrees" rather than "indegrees" or "capacities".

Valid certificate. Notice that all the rules are of the form $(d_1, k_1) \in R_{\alpha_1} \Rightarrow (d, k) \in R_\alpha$ or $(d_1, k_1) \in R_{\alpha_1} \wedge (d_2, k_2) \in R_{\alpha_2} \Rightarrow (d, k) \in R_\alpha$, thus a rule can actually be divided in to two parts: we found a "valid certificate" $(d_1, k_1) \in R_{\alpha_1}$ (and $(d_2, k_2) \in R_{\alpha_2}$, for join nodes), then we add a "product" $(d, k) \in R_\alpha$ based on the certificate. It's easy to see that, every record in R_{α_1} can be a valid certificate in introducing nodes, and every pair of records $((d_1, k_1), (d_2, k_2)) \in R_{\alpha_1} \times R_{\alpha_2}$ can be a valid certificate in join nodes. But in forgetting v nodes, we further require that $d_1(v)$ satisfies some condition. To be specific, in a forgetting node α_1 we say $(d_1, k_1) \in R_{\alpha_1}$ is valid if it satisfies the following condition:

$(\star)$ $\quad d_1(v) = |N(v) \cap Y_\alpha|$ or $|N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|$ for some $\Delta(v) \subseteq N(v) \cap X_\alpha$.

▶ **Theorem 5.** *The set $\{R_\alpha : \alpha \in V(T)\}$ can be computed by the recursive rules above in time $n^{w+O(1)}$, and the **Expected Properties** are satisfied.*

The proof of the correctness of these rules are presented in Appendix A. As $|R_\alpha| \le n^{w+2}$ for all $\alpha \in V(T)$ and the enumerating $\Delta(v)$ procedure in dealing with a forgetting node runs in time $w^{O(w)}$, it's not hard to see that this algorithm runs in time $n^{w+O(1)}$ (for w small enough compared to n).

4 Approximation Algorithm for CVC

Let ϵ be a small value to be determined later. We try to compute an approximate record set $\hat{R}_\alpha$ for each node α, still using bottom-up dynamic programming like what we do in the exact algorithm. An approximate record is a pair $(\hat{d}, \hat{k})$, where $\hat{k} \in \mathbb{N}$ and $\hat{d}$ is a mapping from X_α to $\mathbb{N}_\epsilon = \{0\} \cup \{(1+\epsilon)^x | x \in \mathbb{N}\}$. As we can see, $\hat{d}$ can take non-integer values.

Height of a Node The height h of a node α is defined by the length of the longest path from α to a leaf which is descendent to α. By this definition, the height of a node is 1 plus the maximum height among its children's. Let the height of the root node be h_0. According to the property of nice tree decompositions, h_0 is at most $O(w^2 \log n)$.

Let ϵ_h, δ_h be two variables (which are functions of h, n and w) to be determined later.

***h*-close records.** If an exact record (d, k) and an approximate record $(\hat{d}, \hat{k})$ satisfy $d(v) \sim_{\epsilon_h} \hat{d}(v)$ for all $v \in X_\alpha$ and $k \sim_{\delta_h} \hat{k}$, then we say these two records are h-close.

We expect that for each node α, $\hat{R}_\alpha$ satisfies the following. Let the height of α be h.

(A) If $(d, k) \in R_\alpha$, then there exists $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$ which is h-close to (d, k).

(B) If $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$, then there exists $(d, k) \in R_\alpha$ which is h-close to $(\hat{d}, \hat{k})$.

After $\hat{R}_{\alpha_0}$ is correctly computed (i.e. satisfying (A) and (B)), we output the value $\hat{k}_{\min} = (1 + \delta_{h_0}) \min_{(\hat{d}, \hat{k}) \in \hat{R}_{\alpha_0}} \hat{k}$. Let OPT be the size of the minimum solution, which equals to $\min_{(d,k) \in R_{\alpha_0}} k$. We claim that $\hat{k}_{\min} \in [OPT, (1 + \delta_{h_0})^2 OPT]$.

Proof. By property (B), we have $OPT \le (1 + \delta_{h_0}) \min_{(\hat{d}, \hat{k}) \in \hat{R}_{\alpha_0}} \hat{k}$. By property (A), we have $\min_{(\hat{d}, \hat{k}) \in \hat{R}_{\alpha_0}} \hat{k} \le (1 + \delta_{h_0}) OPT$. The claim follows by combining these two inequalities. ◀

We need the following procedure to test in polynomial time if a sub-problem is solvable when we are allowed to use all vertices to cover the edges.

▶ **Lemma 6.** *Testing whether $(d, |Y_\alpha|) \in R_\alpha$ for any d can be done in $n^{O(1)}$ time.*

Proof. Construct a directed graph with vertex set $\{s,t\} \cup (E[V_\alpha] \setminus E[X_\alpha]) \cup V_\alpha$. For each $e \in (E[V_\alpha] \setminus E[X_\alpha])$ add an edge (s,e) with capacity 1. For each $e = (u,v) \in (E[Y_\alpha] \setminus E[X_\alpha])$ add an edge (e,u) and an edge (e,v) both with capacity 1. For each $v \in X_\alpha$ add an edge (v,t) with capacity $|N(v) \cap Y_\alpha| - d(v)$. For each $v \in Y_\alpha$ add an edge (v,t) with capacity $c(v)$. We claim that $(d, |Y_\alpha|) \in R_\alpha$ if and only if there is a flow from s to t with value $|E[V_\alpha] \setminus E[X_\alpha]|$. For the 'if' part, notice that by the well-known integrality theorem for network flow, there exists a integral flow with the same value. Every integral flow with this value can be transform to an O as expected in the **Expected Properties**: An edge $e \in E[Y_\alpha] \setminus E[X_\alpha]$ is oriented so that it sinks at vertex v if (e,v) has flow value 1, then for each vertex $v \in X_\alpha$, reverse some edges in $E[\{v\}, Y_\alpha]$ so that $D^+_O(v) = d(v)$, if the flow carried in (v,t) is less than $|N(v) \cap Y_\alpha| - d(v)$. One can construct a flow with the value based on an orientation O, too. Thus the 'only if' part is easy to see, too. ◀

We first define $\{\hat{R}_\alpha : \alpha \in V(T)\}$ using the following **Recursive Rules**. Then we prove that these sets satisfy the properties (A) and (B). The basic idea of our approximate algorithm is to run the exact algorithm in an "approximate way". For a rule formed as $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1} \Rightarrow (\hat{d}, \hat{k}) \in \hat{R}_\alpha$ or $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1} \wedge (\hat{d}_2, \hat{k}_2) \in \hat{R}_{\alpha_2} \Rightarrow (\hat{d}, \hat{k}) \in \hat{R}_\alpha$, we also call $(\hat{d}_1, \hat{k}_1)$ (and $(\hat{d}_2, \hat{k}_2)$) the certificate while $(\hat{d}, \hat{k})$ is the product.

Recursive Rules for $\hat{R}_\alpha$

Fix a node $\alpha \in V(T)$ with height h, in case α is a:

Leaf Node. $\hat{R}_\alpha = \{(\hat{d}, \hat{k})\}$, in which $\hat{d}$ is a mapping with empty domain and $\hat{k} = 0$.

Introducing v Node. A record $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$ if and only if $(\hat{d} \setminus v, \hat{k}) \in \hat{R}_{\alpha_1}$ and $\hat{d}(v) = 0$.

Join Node. $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$ if and only if there exists $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1}, (\hat{d}_2, \hat{k}_2) \in \hat{R}_{\alpha_2}$ such that for each $v \in X_\alpha$, $\hat{d}(v) = [\hat{d}_1(v) + \hat{d}_2(v)]_\epsilon$ and $\hat{k} = \hat{k}_1 + \hat{k}_2$.

Forgetting v Node. This case is the most complicated. Let's think this way: we pick $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1}$ and based on it we try to construct $(\hat{d}, \hat{k})$ to add into $\hat{R}_\alpha$. Notice that in the exact algorithm, not every $(d_1, k_1) \in R_{\alpha_1}$ can be used to generate a corresponding product $(d, k) \in R_\alpha$ — it has to be the case that $d_1(v) = |N(v) \cap Y_\alpha|$ or $d_1(v) \geq |N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|$, which is what we called to be a valid certificate. We have to test both the validity of the certificate and its exact counterpart using an indirect way. So there are three issues we need to address:

- **(a)** The requirement for $(\hat{d}_1, \hat{k}_1)$ being valid, i.e. satisfying the "approximate version" of condition $(\star)$;
- **(b)** There exists a valid exact counterpart (d_1, k_1) of $(\hat{d}_1, \hat{k}_1)$ satisfying condition $(\star)$;
- **(c)** How to construct $(\hat{d}, \hat{k})$.

Formally, suppose we have $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1}$, we consider two cases:

- **(1)** v is not "included".
  - (1a) See if $\hat{d}_1(v) \sim_{\epsilon_{h-1}} |N(v) \cap Y_\alpha|$;
  - (1b) See if $(d_t, |Y_{\alpha_1}|) \in R_{\alpha_1}$, where $d_t(u) = \lceil \hat{d}_1(u)/(1+\epsilon_{h-1}) \rceil$ for all $u \in X_{\alpha_1} \setminus \{v\}$ and $d_t(v) = |N(v) \cap Y_\alpha|$ (This is polynomial-time tractable by Lemma 6);
  - (1c) If (a) and (b) are satisfied, then add $(\hat{d}, \hat{k})$ to $\hat{R}_\alpha$, where $\hat{d} = \hat{d}_1 \setminus v, \hat{k} = \hat{k}_1$.
- **(2)** v is "included". We enumerate $\Delta(v) \subseteq N(v) \cap X_\alpha$ and integer A satisfying $A \in [|N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|, |N(v) \cap Y_\alpha|]$.
  - (2a) See if $\hat{d}_1(v) \geq A/(1 + \epsilon_{h-1})$;
  - (2b) See if $(d_t, |Y_{\alpha_1}|) \in R_{\alpha_1}$, where $d_t(u) = \lceil \hat{d}_1(u)/(1+\epsilon_{h-1}) \rceil$ for all $u \in X_{\alpha_1} \setminus \{v\}$ and $d_t(v) = A$ (By Lemma 6, this is still polynomial-time tractable);

(2c) If (a) and (b) are satisfied, then add $(\hat{d}, \hat{k})$ to $\hat{R}_\alpha$, where $\hat{d}(u) = \hat{d}_1(u)$ for all $u \in X_\alpha \setminus \Delta(v)$, $\hat{d}(u) = [\hat{d}(u) + 1]_\epsilon$ for all $u \in \Delta(v)$, $\hat{k} = \hat{k}_1 + 1$.

▶ **Theorem 7.** *Set $\epsilon = \frac{1}{(w^2 \log n)^3}$, $\epsilon_h = 2h\epsilon$ and $\delta_h = 4(h+1)h\epsilon$. Suppose n is large enough. When the dynamic programming is done, for all α, $\hat{R}_\alpha$ satisfies property (A) and (B).*

Proof. (of Theorem 1) According to Theorem 7 and the above discussion, we immediately get $\hat{k}_{\min} \in [OPT, (1+\delta_{h_0})^2 OPT]$. By the property of nice tree decomposition, h_0 is at most $O(w^2 \log n)$, thus $\hat{k}_{\min} \in [OPT, (1 + O(1/(w^2 \log n)))^2 OPT] = [OPT, (1 + O(1/(w^2 \log n)))OPT]$.

The space we need to memorize each $\hat{R}_\alpha$ is $O((w^6 \log^4 n)^w n^{O(1)})$. Computing a leaf/introduce/join node we need $O((w^6 \log^4 n)^{2w} n^{O(1)})$ time. In a forgetting node, we may need to enumerate some set $\Delta(v) \subseteq N(v) \cap X_\alpha$, which requires time $O(2^{|X_\alpha|}) = O(2^{w+1})$. So computing a Forgetting node requires $O((w^6 \log^4 n)^w 2^w n^{O(1)})$ time. The tree size is polynomial, so the total running time is FPT. ◀

To prove Theorem 7, we need a few lemmas. The proof of Lemma 8 and Lemma 9 are presented in Appendix B.

▶ **Lemma 8.** *If $(d, k) \in R_\alpha$ for some node α, then for every (d', k') with $d(v) \geq d'(v)$ for all $v \in X_\alpha$ and k' satisfying $k \leq k' \leq |Y_\alpha|$, we have $(d', k') \in R_\alpha$.*

▶ **Lemma 9.** *Let $a, b, a', b' \in \mathbb{R}, h \in \mathbb{N}^+$, $\epsilon_h \in (0, 0.01)$, $a' \sim_{\epsilon_h} a$ and $b' \sim_{\epsilon_h} b$. Then we have $[a' + b']_\epsilon \sim_{\epsilon_{h+1}} (a + b)$.*

▶ **Lemma 10.** *For all $(d, k) \in R_\alpha$ and $v \in X_\alpha, k \geq d(v)$.*

Proof. Let O be the orientation. Let $N^+(v) = \{u \in V(G) : (v, u) \in E(G)\}$ be v's out neighbor. By definition, we have $d(v) = |N^+(v)| \leq |\{u \in Y_\alpha | D^-_O(u) > 0\}| \leq k$. ◀

▶ **Lemma 11.** *Fix some $(d, k) \in R_\alpha, v \in X_\alpha$ and some integer $p > 0$ satisfying $k + p \leq |Y_\alpha|$. Let $d_m : X_\alpha \to \mathbb{N}$ be a function such that $d_m(v) = d(v) + p$ and $d_m \setminus v = d \setminus v$. We have $(d_m, |Y_\alpha|) \in R_\alpha$ if and only if $(d_m, k + p) \in R_\alpha$.*

Proof. On one hand, the 'if' part is obvious by Lemma 8. On the other hand, we prove that $(d_m, |Y_\alpha|) \in R_\alpha$ implies $(d', k+1) \in R_\alpha$, where $d'(v) = d(v) + 1, d' \setminus v = d \setminus v$. Then we can repeatedly increase the value of k by 1 for p times to obtain the 'only if' part. Let the orientation corresponding to (d, k) and $(d_m, |Y_\alpha|)$ be O_1, O_2 respectively. Now let G' be a graph with vertex set $Y_\alpha \cup \{v\}$. A directed edge (x, y) is in G' if and only if $(x, y) \in O_2$ and $(y, x) \in O_1$.

By picking O_1 so that the number of edges in G' is minimized, we can assume that G' contains no cycle. Otherwise if G' contains a cycle, we can reverse every edge along the cycle in O_1 so that it's still a valid orientation for (d, k) but the number of edges in G' decreases.

As $D^+_{O_2}(v) > D^+_{O_1}(v)$, there exists an non-empty path in G' starting from v ending at, say, $v' \neq v$ such that v' has no out edge in G'. This implies $D^-_{O_1}(v') \leq D^-_{O_2}(v') - 1$, or v' will have an out edge in G'. We reverse the edges along this path in O_1. Let the new orientation be O_3. $D^-_{O_3}(v') \leq D^-_{O_1}(v') + 1 \leq D^-_{O_2}(v') \leq c(v)$. Moreover, $\{u | D^-_{O_3}(u) > 0\} \setminus \{u | D^-_{O_1}(u) > 0\} \subseteq \{v'\}$. Thus, O_3 is a valid orientation for $(d', k+1)$. ◀

Theorem 7 Proof Sketch

Due to space limit, the complete proof is presented in Appendix B.

We use induction on nodes. It's easy to see that leaf nodes satisfy property (A) and (B), because $R_\alpha = \hat{R}_\alpha$ for every leaf node. Fix a node α of height h, by induction, we assume that every node descendent to α satisfies (A) and (B). We only need to prove that α satisfies both (A) and (B). We make a case discussion based on the type of α. The case where α is a forgetting node is the most complicated and requires lemma 10 and 11. The other two types follow Lampis' framework.

To show α satisfies (A), we need to prove the existence of some $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$ for any given $(d, k) \in R_\alpha$ such that $(\hat{d}, \hat{k})$ and (d, k) are h-close. This is done by first picking up the certificate of (d, k), that is the record $(d_1, k_1) \in R_{\alpha_1}$ (or a pair of records in the case α is a join node, we omit join node case in the following sketch) which "produces" (d, k) based on recursive rules for R_α. Then by induction hypothesis, there is an $(h-1)$-close record $(\hat{d}_1, \hat{k}_1)$ in $\hat{R}_{\alpha_1}$. If α is not a forgetting node, then according to recursive rules for $\hat{R}_\alpha$, there exists $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$. We prove that $(\hat{d}, \hat{k})$ and (d, k) are h-close. If α is a forgetting node, then we verify (1b) or (2b) by applying Lemma 8 on (d_1, k_1).

To show α satisfies (B), if α is not a forgetting node, then we pick up and compare some records in a different order: We start from $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$; Then we pick $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1}$ according to recursive rules for $\hat{R}_\alpha$; Next we pick $(d_1, k_1) \in R_{\alpha_1}$ based on induction hypothesis; Finally we find out $(d, k) \in R_\alpha$ using recursive rules for R_α. If α is a forgetting node, suppose the record $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$ is produced by $(\hat{d}_1, \hat{k}_1)$. The main idea is to apply Lemma 11 on $(d_t, |Y_{\alpha_1}|)$, the record verified by (1b) or (2b), and (d_1, k_1), the record $(h-1)$-close to $(\hat{d}_1, \hat{k}_1)$, so as to show the existence of some $(d, k) \in R_\alpha$. At the same time we use Lemma 10 to bound k.

5 Approximation algorithms for TSS and VDS

In this section, we introduce the *vertex subset problem* which is a generalization of many graph problems. Then we present a sufficient condition for the existence of parameterized approximation algorithms for such problems parameterized by the treewidth. Finally, we apply our algorithm to Target Set Selection (TSS) and Vector Dominating Set (VDS), which are both vertex subset problems satisfying this condition. The definitions bellow are inspired by Fomin, et al. [19].

▶ **Definition 12** (Vertex Subset Problem). *A vertex subset problem Φ takes a string $I \in \{0,1\}^*$ as an input, which encodes a graph $G_I = (V_I, E_I)$. Φ is identified by a function $\mathcal{F}_\Phi$ which maps a string $I \in \{0,1\}^*$ to a family of vertex subsets of V_I, say $\mathcal{F}_\Phi(I) \subseteq 2^{V_I}$. Any vertex set in $\mathcal{F}_\Phi(I)$ is a* ***solution*** *of the instance I. The goal is to find a minimum sized solution.*

▶ **Definition 13** (Partial Vertex Subset Problem). *Let Φ be a vertex subset problem. The partial version of Φ takes a string $I \in \{0,1\}^*$ appended with a vertex subset $U \subseteq V_I$ as input. We call such a pair (I, U) a partial instance of Φ. Any vertex set $W \subseteq V_I \setminus U$ is a solution if and only if $W \cup U \in \mathcal{F}_\Phi(I)$. Still, the goal is to find a minimum sized solution.*

We consider the following conditions of a vertex subset problem Φ.

- Φ is **monotone**, if for any instance I, $S \in \mathcal{F}_\Phi(I)$ implies for all S' satisfying $S \subseteq S' \subseteq V_I$, $S' \in \mathcal{F}_\Phi(I)$.
- Φ is **splittable**, if: for any instance I and any separator X of G_I which separates $V_I \setminus X$ into disconnected parts $V_1, V_2, ..., V_p$, if $S_1, S_2, ..., S_p$ are vertex sets such that S_i is a solution for the partial instance $(I, V_I \setminus V_i), \forall 1 \le i \le p$, then $X \cup \bigcup_{1 \le i \le p} S_i$ is a solution for I.

It is trivial to show the monotonicity for Target Set Selection and Vector Dominating Set. To see that they are splittable, observe that given an instance $I = (G, t)$ of Vector Dominating Set for example, fix some $X \subseteq V(G)$, a set S containing X is a solution for I if and only if $S \setminus X$ is a solution for $I' = (G', t')$, where $G' = G[V \setminus X]$ and $t'(v) = t(v) - |N(V) \cap X|$ for all $v \in V \setminus X$. If X is a separator, then the graph G' is not connected, and the union of any solutions of each component in G', with X together forms a solution of I. This observation also works for Target Set Selection.

The main theorem in this section is to show the tractability, in the sense of parameterized approximation, of monotone and splittable vertex subset problems with bounded treewidth.

▶ **Theorem 14.** *Let Φ be a vertex selection problem which is monotone and splittable. If there exists an algorithm such that on input a partial instance of Φ appended with a corresponding nice tree decomposition with width w, it can run in time $f(\ell, w, n)$ and*

- *either output the optimal solution, if the size of it is at most ℓ;*
- *or confirm that the optimal solution size is at least $\ell + 1$*

then there exists an approximate algorithm for Φ with ratio $1 + (w + 1)/(l + 1)$ and runs in time $f(l, w, n) \cdot n^{O(1)}$, for all $l \in \mathbb{N}$.

We provide a trivial algorithm for the partial version of Target Set Selection. Given a partial instance $(I = (G, t), U)$, we search for a solution of size at most ℓ by brute-force. This takes time $f(\ell, w, n) = n^{\ell+O(1)}$. Setting $l := C$ in Theorem 14, we simply get the following.

▶ **Corollary 15.** *(Restated version of Theorem 3) For all constant C, Target Set Selection admits a $1 + (w + 1)/(C + 1)$-approximation algorithm running in time $n^{C+O(1)}$.*

As mentioned before, Raman et al.[29] showed that VDS is $W[1]$-hard parameterized by w, but FPT with respect to the combined parameter $(k + w)$ where k is the solution size. The running time of their algorithm is $k^{O(wk^2)} n^{O(1)}$. A partial instance (I, U) of VDS can be transformed to an equivalent VDS instance, in which the input graph is $G[V_I \setminus U_I]$, so this algorithm can also be used for the partial version of VDS. Set $l := w^2$ in Theorem 14, we get Corollary 16.

▶ **Corollary 16.** *(Restated version of Theorem 2) Vector Dominating Set admits a $1 + (w + 1)/(w^2 + 1)$-approximation algorithm running in time $2^{O(w^5 \log w)} n^{O(1)}$.*

5.1 The Algorithm Framework

To prove Theorem 14, we introduce the concept of l-good node.

▶ **Definition 17** (l-good Node)**.** *Let I be an instance of a vertex selection problem Φ and $(T, \mathcal{X})$ be a nice tree decomposition of any subgraph of G_I. A node $\alpha \in V(T)$ is an l-good node if the partial instance $(I, V_I \setminus Y_\alpha)$ admits a solution of size at most l.*

For a node α, let N_α^- denote the set of all children of α. We post the pseudocode of our algorithm in Algorithm 1. Figure 1 in Appendix C illustrates how the sets defined in Algorithm 1 are related. Algorithm 1 solves the partial version of Φ. For the original version, when we get an instance I, we just create an equivalent partial instance $(I, \emptyset)$ appended with a nice tree decomposition $(T, \mathcal{X})$ and an integer l, then we run *Solve*$((I, \emptyset), (T, \mathcal{X}), l)$. The analyze of Algorithm 1 is presented in Appendix C.

Main idea of Algorithm 1: Let *Alg* be an algorithm solving partial instances in time $f(l, w, n)$. Given a partial instance (I, D) and a nice tree decomposition $(T, \mathcal{X})$ on $G[I \setminus D]$, we run *Alg* to test the goodness of each node. If the root node is l-good, then (I, D) has a solution with size at most l, we use Alg to find the optimal solution. If a leaf node is not l-good then by monotonicity I has no solution[4].Otherwise, we can pick a lowest node α which is not l-good. Then all its children are l-good. Such a node has nice properties.

- On one hand, by the l-goodness of α's children, the partial instances $(I, V_I \setminus Y_{\alpha_c})$ can be optimally solved by Alg for each α_c a child of α. Adding X_{α_c} and the optimal solution E_{α_c} for $(I, V_I \setminus Y_{\alpha_c})$ into the solution enables us to "discard" the whole subtree rooted by α_c and the corresponding vertices, i.e. V_{α_c};
- On the other hand, as α is not l-good, by the splittable and monotone properties, we can deduce that the optimal solution S^* has an $(l+1)$-lower-bounded intersection with Y_α i.e. $|S^* \cap Y_\alpha| \geq l+1$.

Based on these properties, the algorithm iteratively finds one such node α and includes $X_{\alpha_c} \cup E_{\alpha_c}$ for its every child α_c into the solution, then "removes" V_{α_c} from the graph. Once we repeat this procedure, the optimal solution size decreases by at least $|S^* \cap (\bigcup_{\alpha_c} V_{\alpha_c})| \geq |S^* \cap Y_\alpha| \geq l+1$. For each α_c, we use Alg to find the optimal solution E_{α_c}, so in each Y_{α_c} we select at most $|S^* \cap Y_{\alpha_c}|$ vertices. The "non-optimal" part is $\bigcup_{\alpha_c} X_{\alpha_c}$, which is at most $O(w) = O(w/l)|S^* \cap (\bigcup_{\alpha_c} V_{\alpha_c})|$. Therefore, the approximation ratio is upper bounded by $1 + \frac{|\bigcup_{\alpha_c} X_{\alpha_c}|}{l+1} \leq 1 + O(w/l)$.

Algorithm 1 Subprocess $Solve()$

Input: A partial instance (I, D) of Φ, a nice tree decomposition $(T, \mathcal{X})$ of $G_I[V_I \setminus D]$ with width w, $l \in \mathbb{N}$ an integer.

Output: A solution S to (I, D), or 'there exists no solution'.

```
for each node α do
    Use Alg to test if α is an l-good node;
    if α is l-good then
        E_α := the minimum solution for (I, V_I \ Y_α);
    end
end
if the root α_0 is l-good then
    Return E_α0;
end
Find a node α which is not l-good with minimum height;
if α is a leaf node then
    Return 'there exists no solution';
end
E' := ∅;
F := ∅;
for each α_c ∈ N⁻_α do
    E' := E' ∪ E_αc ∪ X_αc;
    F := F ∪ V_αc;
end
Find a nice tree decomposition (T', X') for G_I[V_I \ (D ∪ F)];
Return E' ∪ Solve((I, D ∪ F), (T', X'), l);
```

[4] By our definition of vertex subset problem, the set of solutions can be empty. However any instance of TSS or VDS admits at least one solution which is the whole vertex set.

References

1 Karl R. Abrahamson, Rodney G. Downey, and Michael R. Fellows. Fixed-parameter tractability and completeness IV: on completeness for W[P] and PSPACE analogues. *Ann. Pure Appl. Log.*, 73(3):235–276, 1995. doi:10.1016/0168-0072(94)00034-Z.

2 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

3 Oren Ben-Zwi, Danny Hermelin, Daniel Lokshtanov, and Ilan Newman. Treewidth governs the complexity of target set selection. *Discrete Optimization*, 8(1):87–96, 2011.

4 Nadja Betzler, Robert Bredereck, Rolf Niedermeier, and Johannes Uhlmann. On bounded-degree vertex deletion parameterized by treewidth. *Discrete Applied Mathematics*, 160(1-2):53–60, 2012.

5 Hans L Bodlaender. A tourist guide through treewidth. *Acta cybernetica*, 11(1-2):1, 1994.

6 Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM J. Comput.*, 27(6):1725–1746, 1998. doi:10.1137/S0097539795289859.

7 Moses Charikar, Yonatan Naamad, and Anthony Wirth. On approximating target set selection. In Klaus Jansen, Claire Mathieu, José D. P. Rolim, and Chris Umans, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2016, September 7-9, 2016, Paris, France*, volume 60 of *LIPIcs*, pages 4:1–4:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.APPROX-RANDOM.2016.4.

8 Ning Chen. On the approximability of influence in social networks. *SIAM Journal on Discrete Mathematics*, 23(3):1400–1415, 2009.

9 Wang Chi Cheung, Michel X Goemans, and Sam Chiu-wai Wong. Improved algorithms for vertex cover with hard capacities on multigraphs and hypergraphs. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1714–1726. SIAM, 2014.

10 Julia Chuzhoy and Joseph Naor. Covering problems with hard capacities. *SIAM Journal on Computing*, 36(2):498–515, 2006.

11 Ferdinando Cicalese, Martin Milanič, and Ugo Vaccaro. On the approximability and exact algorithms for vector domination and related problems in graphs. *Discrete Applied Mathematics*, 161(6):750–767, 2013.

12 Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In *Formal Models and Semantics*, pages 193–242. Elsevier, 1990.

13 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

14 Irit Dinur. Mildly exponential reduction from gap 3sat to polynomial-gap label-cover. *Electron. Colloquium Comput. Complex.*, page 128, 2016. URL: https://eccc.weizmann.ac.il/report/2016/128.

15 Michael Dom, Daniel Lokshtanov, Saket Saurabh, and Yngve Villanger. Capacitated domination and covering: A parameterized perspective. In *International Workshop on Parameterized and Exact Computation*, pages 78–90. Springer, 2008.

16 Rodney G Downey and Michael R Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.

17 Andreas Emil Feldmann, Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020.

18 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

19 Fedor V Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. *Journal of the ACM (JACM)*, 66(2):1–23, 2019.

20 Rajiv Gandhi, Eran Halperin, Samir Khuller, Guy Kortsarz, and Aravind Srinivasan. An improved approximation algorithm for vertex cover with hard capacities. *Journal of Computer and System Sciences*, 72(1):16–33, 2006.

21 Martin Grohe. Logic, graphs, and algorithms. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 357–422. Amsterdam University Press, 2008.

22 Sudipto Guha, Refael Hassin, Samir Khuller, and Einat Or. Capacitated vertex covering with applications. In *Symposium on Discrete Algorithms: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, volume 6, pages 858–865. Citeseer, 2002.

23 Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized complexity of generalized vertex cover problems. In *Workshop on Algorithms and Data Structures*, pages 36–48. Springer, 2005.

24 David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146, 2003.

25 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2- ε. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

26 Michael Lampis. Parameterized approximation schemes using graph widths. In *International Colloquium on Automata, Languages, and Programming*, pages 775–786. Springer, 2014.

27 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2181–2200. SIAM, 2020.

28 Pasin Manurangsi and Prasad Raghavendra. A Birthday Repetition Theorem and Complexity of Approximating Dense CSPs. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 78:1–78:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2017/7463`, `doi:10.4230/LIPIcs.ICALP.2017.78`.

29 Venkatesh Raman, Saket Saurabh, and Sriganesh Srihari. Parameterized algorithms for generalized domination. In *International Conference on Combinatorial Optimization and Applications*, pages 116–126. Springer, 2008.

30 Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986.

31 Barna Saha and Samir Khuller. Set cover revisited: Hypergraph cover with hard capacities. In *International Colloquium on Automata, Languages, and Programming*, pages 762–773. Springer, 2012.

32 Jia-Yau Shiau, Mong-Jen Kao, Ching-Chi Lin, and DT Lee. Tight approximation for partial vertex cover with hard capacities. In *28th International Symposium on Algorithms and Computation (ISAAC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

33 Sam Chiu-wai Wong. Tight algorithms for vertex cover with hard capacities on multigraphs and hypergraphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2626–2637. SIAM, 2017.

A Proof of Theorem 5

It is easy to see that $|R_\alpha| \leq n^{O(w)}$ for all α. And one execution of a recursive rule takes time at most polynomial of the size of some R_α. Thus the total running time is $n^{O(w)} \cdot O(w^2 \log n) = n^{O(w)}$.

Now let's prove the correctness. Say a node is consistent if it satisfies the **Expected Properties**. The goal is to show that every node is consistent. We use induction. For records added to leaf nodes, let O just be an empty orientation (as G_α is an empty graph). (1),(2) and (3) in the **Expected Properties** are all satisfied. And by (3), k has to be zero, so no record is missed. Fix a node α, assume that every node descendent to it is consistent. To avoid being misleading, we mean (d, k) is added to R_α by the algorithm when we say $(d, k) \in R_\alpha$, and we say (d, k) is **as expected** if there exists O satisfying the properties. The proof then contains the 'if' part and the 'only if' part. For the 'if' part we have some satisfying O, (d, k) and aim to prove $(d, k) \in R_\alpha$; for the 'only if' part we have $(d, k) \in R_\alpha$ and aim to prove the existence of a satisfying O. We discuss the type of α.

A.1 The 'If' Part

Introducing v Node

Notice that v is an isolated vertex in G_α (which does not contain $E[X_\alpha]$) and $G_{\alpha_1} = G_\alpha[V_\alpha \setminus \{v\}]$. Let O_1 be the orientation for G_{α_1}, where every edge are oriented just the same as that in O. Let (d_1, k_1) be such that $d_1 = d \setminus v, k_1 = k$. It's easy to see that (d_1, k_1) and O_1 satisfy the properties, so $(d_1, k_1) \in R_{\alpha_1}$. As v is an isolated vertex in G_α, $d(v) = D^+_O(v) = 0$, so $(d, k) \in R_\alpha$.

Join Node

Let O_1, O_2 be the orientation for $G_{\alpha_1}, G_{\alpha_2}$ which are consistent to O. By the **Expected Properties**, we have that $d(v) = D^+_O(v) = D^+_{O_1}(v) + D^+_{O_2}(v)$ for all $v \in X_\alpha$, and $|\{v \in Y_{\alpha_1} | D^-_{O_1}(v) > 0\}| + |\{v \in Y_{\alpha_2} | D^-_{O_2}(v) > 0\}| = |\{v \in Y_\alpha | D^-_O(v) > 0\}| \le k \le |Y_\alpha| = |Y_{\alpha_1}| + |Y_{\alpha_2}|$. Let $(d_1, k_1) \in R_{\alpha_1}$ and $(d_2, k_2) \in R_{\alpha_2}$ be the records corresponding to O_1, O_2, where

- $k_1 = \min\{|\{v \in Y_{\alpha_1} | D^-_{O_1}(v) > 0\}| + k - |\{v \in Y_\alpha | D^-_O(v) > 0\}|, |Y_{\alpha_1}|\}$,
- $k_2 = |\{v \in Y_{\alpha_2} | D^-_{O_2}(v) > 0\}| + \min\{0, k - |Y_{\alpha_1}|\}$.

It's easy to see that for all $v \in X_\alpha, d(v) = d_1(v) + d_2(v)$ and $k = k_1 + k_2$, and $(d_1, k_1) \in R_{\alpha_1}, (d_2, k_2) \in R_{\alpha_2}$. Thus $(d, k) \in R_\alpha$.

Forgetting v Node

Notice that $E(G_\alpha) = E(G_{\alpha_1}) \cup E[v, X_\alpha]$. There are two cases. The first case is that $D^-_O(v) = 0$ and $k \le |Y_\alpha| - 1$ (i.e. we can regard v as "not selected"), which means $D^+_O(v) = |N(v)|$. Let O_1 be the orientation for G_{α_1} which is consistent to O. Let $(d_1, k_1) \in R_{\alpha_1}$ be the record corresponding to O_1, where

$$k_1 = k \in [|\{u \in Y_\alpha | D^-_O(u) > 0\}|, |Y_\alpha| - 1] = [|\{u \in Y_{\alpha_1} | D^-_{O_1}(u) > 0\}|, |Y_{\alpha_1}|].$$

For all $u \in X_\alpha \cap N(v)$, the edge (u, v) is oriented so that it sinks at u, so $d(u) = D^+_O(u) = D^+_{O_1}(u) = d_1(u)$. Thus $d = d_1 \setminus v$. And $d_1(v) = D^+_{O_1}(v) = |N(v) \cap Y_\alpha|$. So $(d, k) \in R_\alpha$.

The other case is that $D^-_O(v) > 0$ or $k = |Y_\alpha|$ (i.e. v is "selected"). Let $N^-_O(v)$ be v's in-neighbors in O ($D^-_O(v) = |N^-_O(v)|$). Let $\Delta(v) = N^-_O(v) \cap X_\alpha$. Let (d_1, k_1) be such that $k_1 = k - 1$, for all $u \in \Delta(v), d_1(u) = d(u) - 1$; for all $u \in X_\alpha \setminus \Delta(v), d_1(u) = d(u)$ and $d_1(v) = |Y_\alpha \cap N(v) \setminus N^-_O(v)|$. Let $A := d_1(v)$, then as (d, k) and O satisfy property (2), $A \in [|N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|, |N(v) \cap Y_\alpha|]$. Let O_1 be an orientation for G_{α_1} which is consistent to O. And for a vertex $u \in X_\alpha$, orient the edge (u, v) so that it sinks at v if $u \in \Delta(v)$; at u if $u \notin \Delta(v)$. It's easy to see that $O_1, (d_1, k_1)$ satisfy the properties, notice that if $D^-_O(v) > 0$ then $|\{u \in Y_{\alpha_1} | D^-_{O_1}(u) > 0\}| = |\{u \in Y_\alpha | D^-_O(u) > 0\}| - 1$, if $k = |Y_\alpha|$ then $k_1 = k - 1 = |Y_{\alpha_1}|$. In both cases $|\{u \in Y_{\alpha_1} | D^-_{O_1}(u) > 0\}| \le k \le |Y_{\alpha_1}|$, thus $(d_1, k_1) \in R_{\alpha_1}$. Moreover, based on the recursive rules, it's easy to verify that (d_1, k_1) is a certificate for (d, k). So $(d, k) \in R_\alpha$.

A.2 The 'Only If' Part

Introducing v Node According to recursive rules, there exists $(d_1, k_1) \in R_{\alpha_1}$, where $d_1 = d \setminus v, k_1 = k$. Let O_1 be the orientation for G_{α_1} corresponding to (d_1, k_1). Let O be the orientation for G_α, which is consistent to O_1 (notice that the edge sets of G_α and G_{α_1} are the same). Consider the **Expected Properties**. For all $u \in X_\alpha \setminus \{v\}, d(u) = d_1(u) = D^+_{O_1}(u) = D^+_O(u)$, $d(v) = 0 = D^+_O(v)$; for all $u \in Y_\alpha, D^-_O(u) = D^-_{O_1}(u) \le c(u)$; $k = k_1 \ge |\{u \in Y_{\alpha_1} | D^-_{O_1}(u) > 0\}| = |\{u \in Y_\alpha | D^-_O(u) > 0\}|$. So (d, k) is as expected.

Join Node According to recursive rules, there exists $(d_1, k_1) \in R_{\alpha_1}$ and $(d_2, k_2) \in R_{\alpha_2}$, where $d_1(u) + d_2(u) = d(u)$ for all $u \in X_\alpha$, and $k_1 + k_2 = k$. Notice that $G_{\alpha_1} \cup G_{\alpha_2} = G_\alpha$, and $E(G_{\alpha_1}) \cup E(G_{\alpha_2}) = \emptyset$. Let O_1 and O_2 be the orientations corresponding to (d_1, k_1) and (d_2, k_2) respectively. Let O be the orientation for G_α such that the edges in $E(G_{\alpha_1})$ are oriented as in O_1 and those in $E(G_{\alpha_2})$ are oriented as in O_2. Consider the **Expected Properties**. For all $u \in X_\alpha$, $d(u) = d_1(u) + d_2(u) = D^+_{O_1}(u) + D^+_{O_2}(u) = D^+_O(u)$; for all $u \in Y_{\alpha_1}, D^-_O(u) = D^-_{O_1}(u) \le c(u)$ and for all $u \in Y_{\alpha_2}, D^-_O(u) = D^-_{O_2}(u) \le c(u)$; As $|\{u \in Y_\alpha | D^-_O(u) > 0\}| = |\{u \in Y_{\alpha_1} | D^-_{O_1}(u) > 0\}| + |\{u \in Y_{\alpha_2} | D^-_{O_2}(u) > 0\}|$, $|Y_\alpha| = |Y_{\alpha_1}| + |Y_{\alpha_2}|$ and $k = k_1 + k_2$, we have $|\{u \in Y_\alpha | D^-_O(u) > 0\}| \le k \le |Y_\alpha|$.

Forgetting v Node According to recursive rules, there are two cases.

(1) There exists $(d_1, k_1) \in R_{\alpha_1}$ where $k = k_1$, $d_1(v) = |N(v) \cap Y_\alpha|$ and $d_1 \setminus v = d$. Let the corresponding orientation be O_1. Notice that $E(G_\alpha) \setminus E(G_{\alpha_1}) = N(v) \cap X_\alpha$. Let O be an orientation for G_α, where the edges in $E(G_{\alpha_1})$ are oriented as in O_1 and the edges in $E[v, X_\alpha]$ are oriented so that they all start at v. Then for all $u \in X_\alpha, d(u) = d_1(u) = D^+_{O_1}(u) = D^+_O(u)$; for all $u \in Y_\alpha \setminus v, D^-_O(u) = D^-_{O_1}(u) \le c(u)$ and $D^-_O(v) = 0 \le c(v)$; $k = k_1 \in [|\{u \in Y_{\alpha_1} | D^-_{O_1}(u) > 0\}|, |Y_{\alpha_1}|] = [|\{u \in Y_{\alpha_1} | D^-_O(u) > 0\}|, |Y_\alpha| - 1]$.

(2) There exists $\Delta(v) \subseteq N(v) \cap X_\alpha$, $A \in [|N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|, |N(v) \cap Y_\alpha|]$, and (d_1, k_1) such that $k_1 = k - 1$, $d_1(v) = A$, ... (as described in the recursive rule). Let O_1 be the corresponding orientation. Let O be an orientation for G_α, where the edges in $E(G_{\alpha_1})$ are oriented as in O_1; for an edge (v, u) where $u \in N(v) \cap X_\alpha$, if $u \in \Delta(v)$ then orient the edge so that it sinks at v, otherwise orient it so that it sinks at u. First let's check the indegree of v in O. $D^-_O(v) = D^-_{O_1}(v) + |\Delta(v)| = |N(v) \cap Y_\alpha| - A + |\Delta(v)| \le c(v)$. For all $u \in \Delta(v), d(u) = d_1(u) + 1 = D^+_{O_1}(u) + 1 = D^+_O(u)$ (v becomes its new out-neighbor); for all $u \in X_\alpha \setminus \Delta(v), d(u) = d_1(u) = D^+_{O_1}(u) = D^+_O(u)$; for all $u \in Y_\alpha \setminus \{v\}, D^-_O(u) = D^-_{O_1}(u) \le c(u)$; $|\{u \in Y_\alpha | D^-_O(u) > 0\}| \le |\{u \in Y_{\alpha_1} | D^-_{O_1}(u) > 0\}| + 1$ and $|Y_\alpha| = |Y_{\alpha_1}| + 1$, so $|\{u \in Y_\alpha | D^-_O(u) > 0\}| \le k \le |Y_\alpha|$.

B Proof of Theorem 7

Before the main proof, we prove Lemma 8 and Lemma 9.

Proof. (Lemma 8) Let O be the orientation for (d, k). For each v, we arbitrarily select $d(v) - d'(v)$ out neighbors of v and reverse each edge between one selected neighbor and v. Let the obtained orientation be O_1. We show that O_1 and (d', k') satisfies the properties. (1) and (3) are trivial. To see (2), observe that $D^-_{O_1}(v) \le D^-_O(v)$ for all $v \in Y_\alpha$. ◀

Proof. (Lemma 9) $a' + b' \in [a/(1 + \epsilon_h) + b/(1 + \epsilon_h), a(1 + \epsilon_h) + b(1 + \epsilon_h)]$, that is $(a' + b') \sim_{\epsilon_h} (a + b)$. As $[a' + b']_\epsilon \sim_\epsilon (a' + b')$, we have $[a' + b']_\epsilon \sim_{\epsilon_{h+1}} (a + b)$. ◀

In the following we start the main proof. Leaf nodes satisfy property (A) and (B) since $R_\alpha = \hat{R}_\alpha$ for a leaf node α. Fix a node α of height h, by induction, we assume that every node descendent to α satisfies (A) and (B). Now we prove α satisfies both (A) and (B).

Proof of (A)

Recall that we have some $(d,k) \in R_\alpha$ now and we aim to show the existence of some $(\hat{d},\hat{k}) \in \hat{R}_\alpha$ which is h-close to (d,k). The case for leaf node is trivial. There are three other cases:

Introducing v Node. Suppose α is an introducing v node and α_1 is its child, then we have a certificate $(d_1,k_1) \in R_{\alpha_1}$, where $d_1 = d \setminus v$, $k_1 = k$. By the induction hypothesis, there exists a record $(\hat{d}_1,\hat{k}_1) \in \hat{R}_{\alpha_1}$ which is $(h-1)$-close to (d_1,k_1). By the recursive algorithm for $\hat{R}$, there exists $(\hat{d},\hat{k}) \in \hat{R}_\alpha$, where $\hat{d} \setminus v = \hat{d}_1, \hat{d}(v) = 0$ and $\hat{k} = \hat{k}_1$. Note that for all $u \in X_\alpha \setminus \{v\}, \hat{d}(u) = \hat{d}_1(u) \sim_{\epsilon_{h-1}} d_1(u) = d(u)$, thus we have $\hat{d}(u) \sim_{\epsilon_h} d(u)$. And $\hat{d}(v) = 0 = d(v)$. Since $\hat{k} = \hat{k}_1 \sim_{\delta_{h-1}} k_1 = k$, we get $k \sim_{\delta_h} \hat{k}$. So $(\hat{d},\hat{k})$ is h-close to (d,k).

Join Node. If α is a join node with children α_1 and α_2, then we have a certificate $(d_1,k_1) \in R_{\alpha_1}$ and $(d_2,k_2) \in R_{\alpha_2}$, where for all $v \in X_\alpha, d_1(v) + d_2(v) = d(v)$ and $k_1 + k_2 = k$. By the induction hypothesis, there exist $(\hat{d}_1,\hat{k}_1) \in \hat{R}_{\alpha_1}$ and $(\hat{d}_2,\hat{k}_2) \in \hat{R}_{\alpha_2}$ which are $(h-1)$-close to (d_1,k_1) and (d_2,k_2) respectively. Note that $(\hat{d}_1,\hat{k}_1),(\hat{d}_2,\hat{k}_2)$ is a valid certificate, so there exists $(\hat{d},\hat{k}) \in \hat{R}_\alpha$, where for all $v \in X_\alpha$, $\hat{d}(v) = [\hat{d}_1(v) + \hat{d}_2(v)]_\epsilon$ and $\hat{k} = \hat{k}_1 + \hat{k}_2$. By Lemma 9, for all $v \in X_\alpha$, $\hat{d}(v) \sim_{\epsilon_h} d(v)$ and $\hat{k} \sim_{\delta_h} k$.

Forgetting Node. If α is a forgetting v node with child α_1, then we have a certificate $(d_1,k_1) \in R_{\alpha_1}$ which satisfies one of the following conditions:

(1) $d_1(v) = |N(v) \cap Y_\alpha|, d_1 \setminus v = d$ and $k_1 = k$.

(2) There exist $\Delta(v) \subseteq N(v) \cap X_\alpha$ and $A \in [|N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|, |N(v) \cap Y_\alpha|]$ such that for all $u \in \Delta(v), d_1(u) = d(u) - 1$ and for all $u \in X_{\alpha_1} \setminus (\Delta(v) \cup \{v\}), d_1(u) = d(u), d_1(v) = A$ and $k_1 = k - 1$.

Notice that these two conditions just correspond to the recursive rules with the same index. By the induction hypothesis, there exists an approximate counterpart of the certificate. To be specific, there exists $(\hat{d}_1,\hat{k}_1) \in \hat{R}_{\alpha_1}$ which is $(h-1)$-close to (d_1,k_1). Consider two sub-cases:

Type (1) certificate. As $(\hat{d}_1,\hat{k}_1)$ is $(h-1)$-close to (d_1,k_1) and $d_1(v) = |N(v) \cap Y_\alpha|$, we have $\hat{d}_1(v) \sim_{\epsilon_{h-1}} |N(v) \cap Y_\alpha|$, which means (1a) is satisfied. Let $(d_t, |Y_{\alpha_1}|)$ be the tested pair in (1b). By the definition of $(d_t, |Y_{\alpha_1}|)$, for all $u \in X_{\alpha_1} \setminus \{v\}, d_t(u) = \lceil \hat{d}_1(u)/(1+\epsilon_{h-1}) \rceil \leq \lceil (1+\epsilon_{h-1})d_1(u)/(1+\epsilon_{h-1}) \rceil = d_1(u)$, and $d_t(v) = d_1(v) = |N(v) \cap Y_\alpha|$. Also observe that $k_1 \leq |Y_{\alpha_1}|$. Thus by Lemma 8, $(d_t, |Y_{\alpha_1}|) \in R_{\alpha_1}$, which means (1b) is satisfied. As (1a), (1b) are satisfied, there exists $(\hat{d},\hat{k}) \in \hat{R}_\alpha$, where $\hat{d} = \hat{d}_1 \setminus v, \hat{k} = \hat{k}_1$. Finally, observe that for all $u \in X_\alpha, d(u) = d_1(u) \sim_{\epsilon_{h-1}} \hat{d}_1(u) = \hat{d}(u)$. $k = k_1 \sim_{\delta_{h-1}} \hat{k}_1 = \hat{k}$. So (d,k) and $(\hat{d},\hat{k})$ are h-close.

Type (2) certificate. As $(\hat{d}_1,\hat{k}_1)$ is $(h-1)$-close to (d_1,k_1) and $d_1(v) = A$, we have $\hat{d}_1(v) \geq A/(1+\epsilon_{h-1})$, which means (2a) is satisfied. Let $(d_t, |Y_{\alpha_1}|)$ be the tested pair in (2b), i.e. for all $u \in X_{\alpha_1} \setminus \{v\}, d_t(u) = \lceil \hat{d}_1(u)/(1+\epsilon_{h-1}) \rceil$ and $d_t(v) = A$. Similarly we have that $d_1(u) \geq d_t(u)$ for all $u \in X_\alpha$ while $k_1 \leq |Y_{\alpha_1}|$. Thus by Lemma 8, $(d_t, |Y_{\alpha_1}|) \in R_{\alpha_1}$, which means (2b) is satisfied. As (2a), (2b) are satisfied, there exists $(\hat{d},\hat{k}) \in \hat{R}_\alpha$, where $\hat{d}(u) = [\hat{d}_1(u)+1]_\epsilon$ for all $u \in X_\alpha \setminus \Delta(v)$, $\hat{d}(u) = \hat{d}_1(u)$ for all $u \in \Delta(v)$, and $\hat{k} = \hat{k}_1 + 1$. For each $u \in \Delta(v)$, $d(u) = d_1(u) \sim_{\epsilon_{h-1}} \hat{d}_1(u) = \hat{d}(u)$; for all $u \in X_\alpha \setminus \Delta(v)$, $d(u) \sim_{\epsilon_h} \hat{d}(u)$ by Lemma 9; $k - 1 = k_1 \sim_{\delta_{h-1}} \hat{k}_1 = \hat{k} - 1$ and thus $k \sim_{\delta_h} \hat{k}$. So (d,k) and $(\hat{d},\hat{k})$ are h-close.

Proof of (B)

Now we have some $(\hat{d},\hat{k}) \in \hat{R}_\alpha$ and we aim to show the existence of some $(d,k) \in R_\alpha$ which is h-close to $(\hat{d},\hat{k})$.

Introducing v Node. Suppose α is an introducing v node with α_1 as its child, then by the the recursive rules we have a certificate $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1}$, where $\hat{d}_1 = \hat{d} \setminus v$, $\hat{k}_1 = \hat{k}$. By induction hypothesis, there exists $(d_1, k_1) \in R_{\alpha_1}$ which is $(h-1)$-close to $(\hat{d}_1, \hat{k}_1)$. (d_1, k_1) is a valid certificate, so there exists $(d, k) \in R_\alpha$, where $d \setminus v = d_1, d(v) = 0$ and $k = k_1$. For all $u \in X_\alpha \setminus \{v\}, d(u) = d_1(u) \sim_{\epsilon_{h-1}} \hat{d}_1(u) = \hat{d}(u)$ so $\hat{d}(u) \sim_{\epsilon_h} d(u)$; $\hat{d}(v) = 0 = d(v)$; $k = k_1 \sim_{\delta_{h-1}} \hat{k}_1 = \hat{k}$, so $k \sim_{\delta_h} \hat{k}$.

Join Node. If α is a join node with α_1 and α_2 as its children, then we have a certificate $(\hat{d}_1, \hat{k}_1) \in \hat{R}_\alpha, (\hat{d}_2, \hat{k}_2) \in \hat{R}_{\alpha_2}$, where for all $v \in X_\alpha, [\hat{d}_1(v) + \hat{d}_2(v)]_\epsilon = \hat{d}(v)$ and $\hat{k}_1 + \hat{k}_2 = \hat{k}$. By induction hypothesis, there exist $(d_1, k_1) \in R_{\alpha_1}, (d_2, k_2) \in R_{\alpha_2}$ which are $(h-1)$-close to $(\hat{d}_1, \hat{k}_1)$ and $(\hat{d}_2, \hat{k}_2)$ respectively. Since $(d_1, k_1), (d_2, k_2)$ is a valid certificate, we have there exists $(d, k) \in R_\alpha$, where for all $v \in X_\alpha, d(v) = d_1(v) + d_2(v)$ and $k = k_1 + k_2$. By Lemma 9, for all $v \in X_\alpha, d(v) \sim_{\epsilon_h} \hat{d}(v)$. And $k \sim_{\delta_h} \hat{k}$.

Forgetting v Node. If α is a forgetting v node, then we have a certificate $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1}$ and a tested pair $(d_t, |Y_{\alpha_1}|) \in R_{\alpha_1}$ in (1b) or (2b) with one of the following types:

(1) $\hat{d}_1(v) \sim_{\epsilon_{h-1}} |N(v) \cap Y_\alpha|$; $\hat{d}_1 \setminus v = \hat{d}$; $\hat{k}_1 = \hat{k}$; $d_t(v) = |N(v) \cap Y_\alpha|$;

(2) there exists $\Delta(v) \subseteq N(v) \cap X_\alpha$ and $A \in [|N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|, |N(v) \cap Y_\alpha|]$ such that for all $u \in \Delta(v), \hat{d}(u) = [\hat{d}_1(u) + 1]_\epsilon$; for all $u \in X_{\alpha_1} \setminus \Delta(v) \cup \{v\}, \hat{d}_1(u) = \hat{d}(u); \hat{d}_1(v) \geq A/(1 + \epsilon_{h-1})$; $\hat{k}_1 = \hat{k} - 1$; $d_t(v) = A$.

In both types, for all $u \in X_{\alpha_1} \setminus \{v\}, d_t(u) = \lceil \hat{d}_1(u)/(1 + \epsilon_{h-1}) \rceil$. Notice that these two types just correspond to the recursive rules with the same index. By induction hypothesis, there exists $(d_1, k_1) \in R_{\alpha_1}$ which is $(h-1)$-close to $(\hat{d}_1, \hat{k}_1)$. By the definition of $(h-1)$-closeness we have that for every $u \in X_{\alpha_1} \setminus \{v\}, d_1(u) \geq \lceil \hat{d}_1(u)/(1 + \epsilon_{h-1}) \rceil = d_t(u)$. Consider the two cases:

Type (1) certificate and tested pair. In this case $d_t(v) = |N(v) \cap Y_\alpha|$ and $\hat{d}_1(v) \sim_{\epsilon_{h-1}} |N(v) \cap Y_\alpha|$. Notice that for all $u \in X_{\alpha_1} \setminus \{v\}, d_t(u) = \lceil \hat{d}_1(u)/(1 + \epsilon_{h-1}) \rceil \leq d_1(u)$. Consider the pair (d_t, k_1^*) where $k_1^* = k_1 + |N(v) \cap Y_\alpha| - d_1(v)$. As $(d_1, k_1), (d_t, |Y_{\alpha_1}|) \in R_{\alpha_1}$, by Lemma 8 and 11, we have $(d_t, k_1^*) \in R_{\alpha_1}$. This is a valid certificate as $d_t(v) = |N(v) \cap Y_\alpha|$. So there exists $(d, k) \in R_\alpha$, where $d = d_t \setminus v$ and $k = k_1^*$.

Then we show that (d, k) is h-close to $(\hat{d}, \hat{k})$. Notice that $\hat{k} = \hat{k}_1 \sim_{\delta_{h-1}} k_1, k = k_1^* = k_1 + |N(v) \cap Y_\alpha| - d_1(v)$. As $d_1(v) \sim_{\epsilon_{h-1}} \hat{d}_1(v)$, thus $d_1(v) \geq |N(v) \cap Y_\alpha|/(1 + \epsilon_{h-1})^2$, thus we have that $|N(v) \cap Y_\alpha| - d_1(v) \leq ((1 + \epsilon_{h-1})^2 - 1)d_1(v) \leq 3\epsilon_{h-1}k_1$. Notice that $d_1(v) \leq k_1$ by Lemma 10. So $k \sim_{3\epsilon_{h-1}} k_1 \sim_{\delta_{h-1}} \hat{k}_1 = \hat{k}$. As $(1 + 3\epsilon_{h-1})(1 + \delta_{h-1}) = 1 + (4h + 6)(h-1)\epsilon + 24h(h-1)^2\epsilon^2 \leq 1 + 4h(h+1)\epsilon$, we have $\hat{k} \sim_{\delta_h} k$.

For all $u \in X_\alpha$, we just have $d(u) = d_t(u) \sim_{\epsilon_{h-1}} \hat{d}_1(u) = \hat{d}(u)$.

Type (2) certificate and tested pair. In this case, there exists $\Delta(v) \subseteq N(v) \cap X_\alpha$ and $A \in [|N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|, |N(v) \cap Y_\alpha|]$ such that $d_t(v) = A$. Still we have that for all $u \in X_{\alpha_1} \setminus \{v\}, d_t(u) \leq d_1(u)$. Let $k_1^* := k_1 + \max\{0, A - d_1(v)\}$. As $(d_1, k_1), (d_t, |Y_{\alpha_1}|) \in R_{\alpha_1}$, by Lemma 8 and 11, we have $(d_t, k_1^*) \in R_{\alpha_1}$. This is a valid certificate as $d_t(v) = A$. So there exists $(d, k) \in R_\alpha$, where for all $u \in X_\alpha \setminus \Delta(v), d(u) = d_t(u)$, for all $u \in \Delta(v), d(u) = d_t(u) + 1$ and $k = k_1^* + 1$.

We use the same idea to show $\hat{k} \sim_{\delta_h} k$. Still, we have $k_1 \geq d_1(v) \geq A/(1 + \epsilon_{h-1})^2$. So $k_1^* = k_1 + \max\{0, A - d_1(v)\} \leq 3\epsilon_{h-1}k_1$ and obviously, $k_1^* \geq k_1$. So $k_1^* \sim_{3\epsilon_{h-1}} k_1$. As $\hat{k} - 1 = \hat{k}_1 \sim_{\delta_{h-1}} k_1$, we have $\hat{k} - 1 \sim_{\delta_h} k_1^* = k - 1$. Thus $\hat{k} \sim_{\delta_h} k$.

For all $u \in X_\alpha \setminus \Delta(v)$, we have $d(u) = {d_1}^*(u) \sim_{\epsilon_{h-1}} \hat{d}_1(u) = \hat{d}$. For all $u \in \Delta(v)$, we have $d(u) - 1 = {d_1}^*(u) \sim_{\epsilon_{h-1}} \hat{d}_1(u)$ and $\hat{d}(u) = [\hat{d}_1(u) + 1]_\epsilon$, by Lemma 9 we have $d(u) \sim_{\epsilon_h} \hat{d}(u)$.

C Proof of Theorem 14

We first prove that for any bag X_α in a tree decomposition for a graph $G=(V,E)$, vertex sets Y_α and $V \setminus V_\alpha$ are disconnected in $G[V\setminus X_\alpha]$ i.e. X_α separates $V\setminus X_\alpha$ into two disconnected parts Y_α and $V\setminus V_\alpha$. Assume they are connected, then there exists $u \in Y_\alpha$ and $v \in V\setminus V_\alpha$ such that $(u,v)\in E$. So there exists some bag containing both u and v. This implies that the nodes whose assigned bags containing u or v forms a subtree in the tree decomposition. However, X divides apart some nodes whose assigned bags containing u or v, a contradiction.

Since $(T,\mathcal{X})$ is a tree decomposition for $G_I[V_I\setminus D]$, a corollary is that for any bag $X_\alpha \in \mathcal{X}$, $X_\alpha \cup D$ separates $V_I \setminus (D\cup X_\alpha)$ into disconnected parts Y_α and $V_I\setminus (V_\alpha \cup D)$.

Now let's analyze Algorithm 1. We use induction. Firstly let's consider basic cases. If (I,D) has a minimum solution of size at most l, then the algorithm returns at line 8 an optimal solution. If (I,D) contains no solution, which is equivalent to V_I is not a solution due to monotonicity, then any leaf node is not l-good since $Y_{\alpha'}=\emptyset$ for a leaf node α' and the algorithm returns at line 12. So in these cases, the algorithm is correct. In the remaining case, the algorithm picks a node α which is not l-good at line 10, then it adds some vertices to the final output and creates a new instance to make a recursive call. Since α is the node which is not l-good node with minimum height, its children are all l-good. Let the optimal solution for (I,D) be S^*. Let $S := \mathit{Solve}((I,D\cup F),(T',\mathcal{X}),l)$ and let S' denote the optimal solution for $(I,D\cup F)$.

▶ **Lemma 18.** *As the problem is monotone and splittable, we have the following:*

(i) *$S^*\cap Y_\alpha$ is a solution for $(I,V_I\setminus Y_\alpha)$.*

(ii) *For all α_c a child of α, $S^*\cap Y_{\alpha_c}$ is a solution for $(I,V_I\setminus Y_{\alpha_c})$;*

(iii) *$S^*\setminus F$ is a solution for $(I,D\cup F)$;*

(iv) *$E'\cup S$ is a solution for (I,D).*

Proof. **(i)** By the definition of partial instances, $S^*\cup D$ is a solution for I. By monotonicity, $S^*\cup D\cup(V_I\setminus Y_\alpha) = S^*\cap Y_\alpha \cup (V_I\setminus Y_\alpha)$ is also a solution for I. So $S^*\cap Y_\alpha$ is a solution for $(I,V_I\setminus Y_\alpha)$ according to the definition of partial solution.

(ii) Similarly as above, by monotonicity, $S^*\cup D\cup(V_I\setminus Y_{\alpha_c}) = S^*\cap Y_{\alpha_c}\cup(V_I\setminus Y_{\alpha_c})$ is also a solution for I. So $S^*\cap Y_{\alpha_c}$ is a solution for $(I,V_I\setminus Y_{\alpha_c})$.

(iii) By monotonicity, $S^*\cup D\cup F$ is also a solution for I. So $S^*\setminus F$ is a solution for $(I,D\cup F)$.

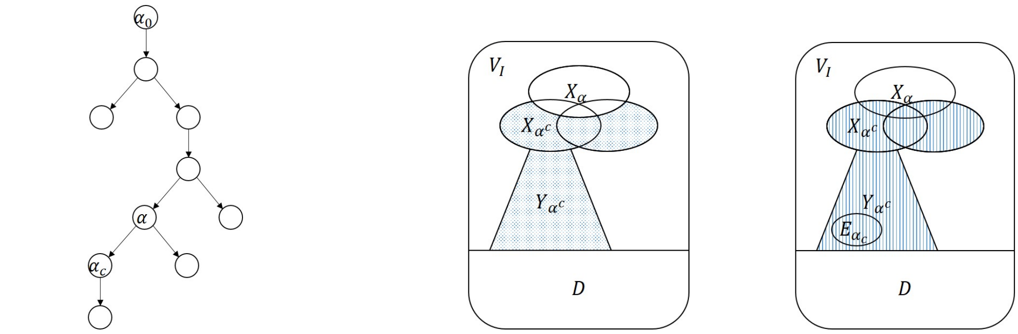
(iv) We need to use the property that Φ is splittable. By the algorithm, $E' = \bigcup_{\alpha_c\in N_\alpha^-} X_{\alpha_c} \cup \bigcup_{\alpha_c\in N_\alpha^-} E_{\alpha_c}$ and $F = \bigcup_{\alpha_c\in N_\alpha^-} V_{\alpha_c}$. Let X' denote $\cup_{\alpha_c\in N_\alpha^-} X_{\alpha_c}$. To use the property that Φ is splittable, observe that $D\cup X'$ is a separator. Each Y_{α_c} is an isolated part (not connected to the remaining graph) in $G_I[V_I\setminus(D\cup X')]$. The remaining part in $G_I[V_I\setminus(D\cup X')]$ is thus isolated and it's $V_I\setminus(D\cup X'\cup\bigcup_{\alpha_c\in N_\alpha^-} Y_{\alpha_c}) = V_I\setminus(D\cup F)$. Because each E_{α_c} is a solution for $(I,V_I\setminus Y_{\alpha_c})$, and by induction hypothesis, S is a solution for $(I,D\cup F)$, we get that Φ is splittable implies $X'\cup D\cup S\cup\bigcup_{\alpha_c\in N_\alpha^-} E_{\alpha_c} = E'\cup D\cup S$ is a solution for I. So $E'\cup S$ is a solution for (I,D).

◀

By induction we assume that $|S|\le(1+(w+1)/(l+1))|S'|$. The approximation ratio is

$$\frac{|S\cup E'|}{|S^*|} \le \frac{|S| + \sum_{\alpha_c\in N_\alpha^-}|E_{\alpha_c}| + |\bigcup_{\alpha_c\in N_\alpha^-} X_{\alpha_c}|}{|S^*\cap F| + |S^*\setminus F|}.$$

Since $|S|/|S^*\setminus F| \le |S|/|S'| \le 1+(w+1)/(l+1)$, we only need to show $(\sum_{\alpha_c\in N_\alpha^-}|E_{\alpha_c}| + |\bigcup_{\alpha_c\in N_\alpha^-} X_{\alpha_c}|)/|S^*\cap F| \le 1+(w+1)/(l+1)$. Notice that by the definition, $Y_\alpha\subseteq F$. Since



(a) T in a tree decomposition $(T, \mathcal{X})$.

(b) The vertex sets about α and α_c. Dotted part is Y_α

(c) E_{α_c} is added, and F is the lined part.

Figure 1 Venn diagram of sets defined in Algorithm 1

α is not l-good, (i) implies that $|S^* \cap F| \geq |S^* \cap Y_\alpha| \geq l+1$. By (ii), for all $\alpha_c \in N_\alpha^-$, $|E_{\alpha_c}| \leq |S^* \cap Y_{\alpha_c}|$. We have

$$\begin{aligned}
&\frac{\sum_{\alpha_c \in N_\alpha^-} |E_{\alpha_c}| + |\bigcup_{\alpha_c \in N_\alpha^-} X_{\alpha_c}|}{|S^* \cap F|} \\
=&\frac{\sum_{\alpha_c \in N_\alpha^-} |E_{\alpha_c}|}{|S^* \cap F|} + \frac{|\bigcup_{\alpha_c \in N_\alpha^-} X_{\alpha_c}|}{|S^* \cap F|} \\
\leq&\frac{\sum_{\alpha_c \in N_\alpha^-} |E_{\alpha_c}|}{\sum_{\alpha_c \in N_\alpha^-} |S^* \cap Y_{\alpha_c}|} + \frac{|\bigcup_{\alpha_c \in N_\alpha^-} X_{\alpha_c}|}{|S^* \cap F|} \text{ (Y_{α_c}'s are disjoint subsets of F)} \\
\leq&1 + \frac{|\bigcup_{\alpha_c \in N_\alpha^-} X_{\alpha_c}|}{|S^* \cap F|} \text{ (By (ii) and the definition of E_{α_c})} \\
\leq&1 + \frac{|\bigcup_{\alpha_c \in N_\alpha^-} X_{\alpha_c}|}{l+1} \text{ (By $|S^* \cap F| \geq l+1$).}
\end{aligned}$$

In a nice tree decomposition, the only case that $|N_\alpha^-| > 1$ is that α is a join node, however in this case, the bags of its two children are the same. So $|\bigcup_{\alpha_c \in N_\alpha^-} X_{\alpha_c}|/(l+1) + 1 \leq (w+1)/(l+1) + 1$. The approximation ratio follows. It's easy to see the algorithm makes at most $n^{O(1)}$ recursive calls, so the running time is $f(l, w, n)n^{O(1)}$. And thus Theorem 14 is proved.