Knapsack with Vertex Cover, Set Cover, and Hitting Set

Palash Dey
Indian Institute of Technology Kharagpur, India

Ashlesha Hota
Indian Institute of Technology Kharagpur, India

Sudeshna Kolay
Indian Institute of Technology Kharagpur, India

Sipra Singh
Indian Institute of Technology Kharagpur, India

Abstract

In the Vertex Cover Knapsack problem, we are given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with weights $(w(u))_{u\in\mathcal{V}}$ and values $(\alpha(u))_{u\in\mathcal{V}}$ of the vertices, the size s of the knapsack, a target value p, and the goal is to compute if there exists a vertex cover $\mathcal{U} \subseteq \mathcal{V}$ with total weight at most s, and total value at least p. This problem simultaneously generalizes the classical vertex cover and knapsack problems. We show that this problem is strongly NP-complete. However, it admits a pseudo-polynomial time algorithm for trees. In fact, we show that there is an algorithm that runs in time $\mathcal{O}\left(2^{\mathrm{tw}} \cdot n \cdot \min\{s^2, p^2\}\right)$ where tw is the treewidth of $\mathcal{G}$. Moreover, we can compute a $(1-\varepsilon)$-approximate solution for maximizing the value of the solution given the knapsack size as input in time $\mathcal{O}\left(2^{\mathrm{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v\in\mathcal{V}} \alpha(v)\right))\right)$ and a $(1+\varepsilon)$-approximate solution to minimize the size of the solution given a target value as input, in time $\mathcal{O}\left(2^{\mathrm{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v\in\mathcal{V}} w(v)\right))\right)$ for every $\varepsilon > 0$. Restricting our attention to polynomial-time algorithms only, we then consider polynomial-time algorithms and present a 2 factor polynomial-time approximation algorithm for this problem for minimizing the total weight of the solution, which is optimal up to additive $o(1)$ assuming Unique Games Conjecture (UGC). On the other hand, we show that there is no ρ factor polynomial-time approximation algorithm for maximizing the total value of the solution given a knapsack size for any $\rho > 1$ unless P = NP.

Furthermore, we show similar results for the variants of the above problem when the solution $\mathcal{U}$ needs to be a minimal vertex cover, minimum vertex cover, and vertex cover of size at most k for some input integer k. Then, we consider set families (equivalently hypergraphs) and study the variants of the above problem when the solution needs to be a set cover and hitting set. We show that there are H_d and f factor polynomial-time approximation algorithms for Set Cover Knapsack where d is the maximum cardinality of any set and f is the maximum number of sets in the family where any element can belongs in the input for minimizing the weight of the knapsack given a target value, and a d factor polynomial-time approximation algorithm for d-Hitting Set Knapsack which are optimal up to additive $o(1)$ assuming UGC. On the other hand, we show that there is no ρ factor polynomial-time approximation algorithm for maximizing the total value of the solution given a knapsack size for any $\rho > 1$ unless P = NP for both Set Cover Knapsack and d-Hitting Set Knapsack.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Knapsack, vertex cover, minimal vertex cover, minimum vertex cover, hitting set, set cover, algorithm, approximation algorithm, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.27

Related Version *Full Version*: https://arxiv.org/abs/2406.01057 [8]

1 Introduction

A *vertex cover* of an undirected graph is a set of vertices that contains at least one endpoint of every edge. For a real-world application of vertex cover, consider a city network $\mathcal{G}$ where the vertices are the major localities of the city, and we have an edge between two vertices if the distance between their corresponding locations is less than, say, five kilometers. A retail chain wants to open a few stores in the city in such a way that everyone (including the people living between any two major localities) in the city has a retail shop within five kilometers. The cost of opening a store depends on location. We can see that the company needs to compute a minimum weight vertex cover of the $\mathcal{G}$ to open stores with the minimum total cost, where the weight of a vertex is the cost of opening a store at that location. However, each store has the potential to generate non-core revenue, say from advertising. In such a scenario, the company may like to maximize the total non-core revenue without compromising its core business, which it will accomplish by opening stores at the vertices of a vertex cover. This is precisely what we call Vertex Cover Knapsack. In this problem, we are given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with weights $(w(u))_{u\in\mathcal{V}}$ and values $(\alpha(u))_{u\in\mathcal{V}}$ of the vertices, the size s of the knapsack, a target value p, and the goal is to compute if there exists a vertex cover $\mathcal{U} \subseteq \mathcal{V}$ with $w(\mathcal{U}) = \sum_{u\in\mathcal{U}} w(u) \leqslant s$, and $\alpha(\mathcal{U}) = \sum_{u\in\mathcal{U}} \alpha(u) \geqslant p$.

We study several natural variations of this problem: (i) k-Vertex Cover Knapsack where the solution should be a vertex cover of size at most k for an integer input k, (ii) Minimal Vertex Cover Knapsack where the solution should be a minimal vertex cover, and (iii) Minimum Vertex Cover Knapsack where the solution should be a minimum vertex cover.

We then consider the hypergraphs or equivalently set families. There, we consider the knapsack generalization of the set cover and hitting set problems. In Set Cover Knapsack, we are given a collection $S_1, \ldots, S_m$ of subsets of a universe $[n]$, with weights $(w(j))_{j\in[m]}$ and values $(\alpha(j))_{j\in[m]}$ for the sets, the size s of the knapsack, a target value p, and the goal is to compute if there exists a set cover of total weight at most s and total value at least p. On the other hand, we have a collection $S_1, \ldots, S_m$ of d sized subsets of a universe $[n]$ with weights in d-Hitting Set Knapsack $(w(j))_{j\in[n]}$ and values $(\alpha(j))_{j\in[n]}$ for the elements, the size s of the knapsack, a target value p, and the goal is to compute if there exists a hitting set of total weight at most s and total value at least p.

1.1 Contributions

We study these problems under the lens of classical complexity theory, parameterized complexity, polynomial-time approximation, and FPT-approximation. We summarize our results in Table 1.

We now give a high-level overview of the techniques used in our results. For the f-approximation algorithm for Set Cover Knapsack, the dual LP of a configuration LP relaxation has two types of constraints: intuitively speaking, one set of constraints handles the knapsack part while the other set takes care of the set cover requirement. We first increase some dual variables iteratively so that some of the dual constraints corresponding to the knapsack part of the problem become tight. We pick the sets corresponding to these constraints. If this gives a valid set cover, then we are done. Otherwise, we increase some dual constraints iteratively corresponding to the set cover part of the problem until we satisfy the set cover requirements. The first part of our H_d-approximation algorithm is the same as the f-approximation algorithm. In the second part, we use the greedy algorithm for the set cover problem to pick more sets if the sets picked in the first part do not form a set cover.

Table 1 Summary of results. tw : treewidth of the graph, s : size of knapsack, p : target value of knapsack, ε : any real number greater than zero, n : number of vertices or size of the universe, f : the maximum number of sets where any element belongs, d : maximum size of any set, ρ : any poly-time computable function. $\star$: size of knapsack is input. † : bag size is input. ‡ : target value is input.

Knapsack variant	Results
Vertex Cover	• Strongly NP-complete (Observation 4) • NP-complete for star graphs (Observation 8) • Poly-time 2-approx. to minimize weight† (Corollary 17) • Poly-time ρ-inapprox. to maximize value* (Theorem 20) • $\mathcal{O}\left(2^{\text{tw}} \cdot n^{\mathcal{O}(1)} \cdot \min\{s^2, p^2\}\right)$ (Theorem 21) • $\mathcal{O}\left(2^{\text{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v\in\mathcal{V}} \alpha(v)\right))\right)$ time, $(1-\varepsilon)$ approximation to maximize value* (Corollary 26) • $\mathcal{O}\left(2^{\text{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v\in\mathcal{V}} w(v)\right))\right)$ time, $(1+\varepsilon)$ approximation to minimize weight‡ (Theorem 27)
Vertex Cover of size $\leqslant k$	• Strongly NP-complete (Corollary 5) • NP-complete for star graphs (Observation 9) • $\mathcal{O}\left(2^{\text{tw}} \cdot n^{\mathcal{O}(1)} \cdot \min\{s^2, p^2\}\right)$ (Theorem 23) • $\mathcal{O}\left(2^{\text{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v\in\mathcal{V}} \alpha(v)\right))\right)$ time, $(1-\varepsilon)$ approximation to maximize value* (Corollary 26) • $\mathcal{O}\left(2^{\text{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v\in\mathcal{V}} w(v)\right))\right)$ time, $(1+\varepsilon)$ approximation to minimize weight‡ (Theorem 27)
Minimum Vertex Cover	• NP hard (Observation 7) • NP-complete for trees (Observation 11) • $\mathcal{O}\left(2^{\text{tw}} \cdot n^{\mathcal{O}(1)} \cdot \min\{s^2, p^2\}\right)$ (Theorem 22) • $\mathcal{O}\left(2^{\text{tw}} \cdot poly(n, 1/\varepsilon, , \log\left(\sum_{v\in\mathcal{V}} \alpha(v)\right))\right)$ time, $(1-\varepsilon)$ approximation to maximize value* (Corollary 26) • $\mathcal{O}\left(2^{\text{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v\in\mathcal{V}} w(v)\right))\right)$ time, $(1+\varepsilon)$ approximation to minimize weight‡ (Theorem 27)
Minimal Vertex Cover	• Strongly NP-complete (Observation 6) • NP-complete for trees (Theorem 10) • No poly-time approx. algorithm neither to maximize value* nor to minimize weight† (Theorem 20) • $\mathcal{O}\left(16^{\text{tw}} \cdot n^{\mathcal{O}(1)} \cdot \min\{s^2, p^2\}\right)$ (Theorem 24) • $\mathcal{O}\left(16^{\text{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v\in\mathcal{V}} \alpha(v)\right))\right)$ time, $(1-\varepsilon)$ approximation to maximize value* (Theorem 25) • $\mathcal{O}\left(16^{\text{tw}} \cdot poly(n, 1/\varepsilon, \log\left(\sum_{v\in\mathcal{V}} w(v)\right))\right)$ time, $(1+\varepsilon)$ approximation to minimize weight‡ (Theorem 27)
Set Cover	• Strongly NP-complete (Observation 12) • Poly-time f-approx. to minimize weight† (Theorem 15) • Poly-time H_d-approx. to minimize weight† (Theorem 18) • Poly-time ρ-inapprox. to maximize value* (Theorem 20)
d-Hitting Set	• Strongly NP-complete (Observation 13) • Poly-time d-approx. to minimize weight† (Corollary 16) • Poly-time ρ-inapprox. to maximize value* (Theorem 20)

Our fixed-parameter pseudo-polynomial time algorithms with respect to treewidth for the variants of vertex cover knapsack combine the idea of pseudo-polynomial time algorithm and the dynamic programming algorithm for vertex cover with respect to treewidth. Then, we use these algorithms in a black-box fashion to obtain FPT-approximation algorithms.

1.2 Related Work

The classical knapsack problem admits a fully polynomial time approximation scheme (FPTAS) [25, 27]. Since our paper focuses on generalizations of knapsack to some graph theoretic problems and their extension to hypergraphs, we discuss only those knapsack variants directly related to ours.

Yamada et al. [28] proposed heuristics for knapsack when there is a graph on the items, and the solutions need to be an independent set. Many intractability results in special graph classes and heuristic algorithms based on pruning, dynamic programming, etc. have been studied for this independent set knapsack problem [18, 19, 23, 1, 17, 5, 21, 24, 14, 13, 2, 22]. Dey et al. [9] studied the knapsack problem with graph-theoretic constraints like - connectedness, paths, and shortest path.

Note that our Vertex Cover Knapsack also generalizes the classical weighted vertex cover problem, for which we know a polynomial-time 2-approximation algorithm which is the best possible approximation factor up to additive $\varepsilon > 0$ that one can achieve in polynomial time assuming Unique Games Conjecture [25, 27]. On the parameterized side, there is a long line of work on designing a fast FPT algorithm for vertex cover parameterized by the size k of a minimum vertex cover, with the current best being $\mathcal{O}\left(1.25284^k \cdot n^{\mathcal{O}(1)}\right)$ [16]. Later, Peter Damaschke [7] proved that it is solvable in time $\mathcal{O}\left(1.62^k \cdot n^{\mathcal{O}(1)}\right)$. Boria et al. [4] showed that there is a polynomial time $n^{-1/2}$ approximation algorithm and inapproximable within the ratio $n^{\varepsilon-1/2}$ in polynomial time unless P = NP, where $\varepsilon > 0$.

Various approximation algorithms have been studied for the Set Cover problem with approximation ratios f where f is the maximum number of sets that any element can belong and H_d where d is the maximum cardinality of any set, and H_d is the d-th harmonic number. These approximation factors are tight up to additive $\varepsilon > 0$ under standard complexity-theoretic assumptions [10, 20, 27, 25].

2 Preliminaries

We denote the set $\{1, 2, \ldots\}$ of natural numbers with $\mathbb{N}$. For any integer ℓ, we denote the sets $\{1, \ldots, \ell\}$ and $\{0, 1, \ldots, \ell\}$ by $[\ell]$ and $[\ell]_0$ respectively. We now define our problems formally. Our first problem is Vertex Cover Knapsack, where we need to find a vertex cover that fits the knapsack and meets a target value. A *vertex cover* of a graph is a subset of vertices that includes at least one end-point of every edge. Formally, it is defined as follows.

▶ **Definition 1** (Vertex Cover Knapsack)**.** *Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, weight of vertices $(w(u))_{u\in\mathcal{V}}$, value of vertices $(\alpha(u))_{u\in\mathcal{V}}$, size s of knapsack, and target value p, compute if there exists a vertex cover $\mathcal{U} \subseteq \mathcal{V}$ of $\mathcal{G}$ with weight $w(\mathcal{U}) = \sum_{u\in\mathcal{U}} w(u) \leqslant s$ and value $\alpha(\mathcal{U}) = \sum_{u\in\mathcal{U}} \alpha(u) \geqslant p$. We denote an any instance of it by $(\mathcal{G}, (w(u))_{u\in\mathcal{V}}, (\alpha(u))_{u\in\mathcal{V}}, s, p)$.*

The k-Vertex Cover Knapsack, Minimum Vertex Cover Knapsack, Minimal Vertex Cover Knapsack problems are the same as Definition 1 except we require the solution $\mathcal{U}$ to be respectively a vertex cover of size at most k for an input integer k, a minimum cardinality vertex cover, a minimal vertex cover.

The treewidth of a graph quantifies the tree likeness of a graph [6]. Informally speaking, a tree decomposition of a graph is a tree where every node of the tree corresponds to some subsets of vertices, called bags, and the tree should satisfy three properties: (i) every vertex of the graph should belong to some bag, (ii) both the endpoints of every edge should belong to some bag, and (iii) the set of nodes of the tree containing any vertex should be connected. We refer to [8] for the formal definition of a tree decomposition, a nice tree decomposition, and the treewidth of a graph.

Extending the notion of vertex cover to hypergraphs, we define the Set Cover Knapsack problem where we need to compute a set cover that fits the knapsack and achieves a maximum value. Formally, we define it as follows.

▶ **Definition 2** (Set Cover Knapsack). *Given a collection $\mathcal{F} = \{S_1, \ldots, S_m\}$ of subsets of a universe $[n]$ with weights $(w_j)_{j\in[m]}$ and values $(\alpha_j)_{j\in[m]}$ of the sets, size s of knapsack, and target value p, compute if there exists a set cover $\mathcal{J} \subseteq [m]$ of weight $w(\mathcal{J}) = \sum_{j\in\mathcal{J}} w_j \leqslant s$ and value $\alpha(\mathcal{J}) = \sum_{j\in\mathcal{J}} \alpha_j \geqslant p$. We denote any instance of it by $([n], \mathcal{F}, (w_j)_{j\in[m]}, (\alpha_j)_{j\in[m]}, s, p)$.*

We also define d-Hitting Set Knapsack, where we need to compute a hitting set that fits the knapsack and achieves at least the target value; here, items have weights and values.

▶ **Definition 3** (d-Hitting Set Knapsack). *Given a collection $\mathcal{F} = \{S_1, \ldots, S_m\}$ of subsets of a universe $[n]$ of size at most d with weights $(w_i)_{i\in[n]}$ and values $(\alpha_i)_{i\in[n]}$ of the items, size s of knapsack, and target value p, compute if there exists a hitting set $\mathcal{I} \subseteq [n]$ of weight $w(\mathcal{I}) = \sum_{i\in\mathcal{I}} w_i \leqslant s$ and value $\alpha(\mathcal{I}) = \sum_{i\in\mathcal{I}} \alpha_i \geqslant p$. We denote any instance of it by $([n], \mathcal{S}, (w_i)_{i\in[n]}, (\alpha_i)_{i\in[n]}, s, p)$.*

If not mentioned otherwise, we use n to denote the number of vertices for problems involving graphs and the size of the universe for problems involving a set family; m to indicate the number of edges for problems involving graphs and the number of sets in the family of sets for problems involving a set family; tw to denote the treewidth of the graph; s to represent the size of knapsack, and p to denote the target value of solution.

3 Results: Classical NP Completeness

In this section, we present our NP-completeness results. Our first results show that Vertex Cover Knapsack is strongly NP-complete, that is, it is NP-complete even if the weight and value of every vertex are encoded in unary. We reduce from the classical Vertex Cover problem, where the goal is to find a vertex cover of cardinality at most some input integer k. Vertex Cover is known to be NP-complete even for 3 regular graphs [11, folklore]. To reduce a Vertex Cover instance to a Vertex Cover Knapsack instance, we define the weight and value of every vertex to be 1, and the size and target value to be k. In the interest of space, we omit the proofs of a few of our results. They are marked ($\star$). We refer to [8] for the detailed algorithm with proof of correctness and the analysis of its running time.

▶ **Observation 4.** Vertex Cover Knapsack *is strongly* NP*-complete.*

The same reduction in Observation 4 also shows that k-Vertex Cover Knapsack is strongly NP-complete.

▶ **Corollary 5.** k-Vertex Cover Knapsack *is strongly* NP*-complete.*

In the Maximum Minimal Vertex Cover problem, the goal is to compute if there exists a minimal vertex cover of cardinality at least some input integer. A vertex cover of a graph is called minimal if no proper subset of it is a vertex cover. Maximum Minimal Vertex Cover is known to be NP-complete [15, 3]. We show that the same reduction as in the proof of Observation 4 except starting from an instance of Maximum Minimal Vertex Cover instead of Vertex Cover, proves that Minimal Vertex Cover Knapsack is strongly NP-complete.

▶ **Observation 6** ($\star$). Minimal Vertex Cover Knapsack *is strongly* NP*-complete.*

We show similar results for Minimum Vertex Cover Knapsack except that it does not belong to NP unless the polynomial hierarchy collapses.

▶ **Observation 7** (⋆). Minimum Vertex Cover Knapsack *is strongly* NP*-hard.*

We show next that Vertex Cover Knapsack is NP-complete even if the underlying graph is a tree by reducing it from the classical Knapsack– simply add the knapsack items as leaves of a star graph. However, it turns out that they are not strongly NP-complete for trees. We will see in Section 5 that they admit pseudo-polynomial time algorithms for trees.

▶ **Observation 8** (⋆). Vertex Cover Knapsack *is* NP*-complete for star graphs.*

By setting k to be the number of leaves, the reduction in the proof of Observation 8 also shows NP-completeness for k-Vertex Cover Knapsack.

▶ **Observation 9** (⋆). k-Vertex Cover Knapsack *is* NP*-complete for star graphs.*

Note that the reduction from Knapsack to Vertex Cover Knapsack for star graphs does not work for Minimal Vertex Cover Knapsack and Minimum Vertex Cover Knapsack. Indeed, for star graphs, both the problems admit polynomial-time algorithms. Nevertheless, we are able to show that both the problems are (not strongly) NP-complete for trees.

▶ **Theorem 10** (⋆). Minimal Vertex Cover Knapsack *is* NP*-complete for trees.*

▶ **Observation 11** (⋆). Minimum Vertex Cover Knapsack *is* NP*-complete for trees.*

Note that, since the size of a minimum vertex cover in a tree can be computed in polynomial time thanks to König's Theorem [26], Minimum Vertex Cover Knapsack belongs to NP.

We show similar results for Set Cover Knapsack and d-Hitting Set Knapsack also by reducing from respectively unweighted set cover and unweighted d-hitting set, both of which are known to be NP-complete [12].

▶ **Observation 12** (⋆). Set Cover Knapsack *is strongly* NP*-complete.*

▶ **Observation 13** (⋆). d-Hitting Set Knapsack *is strongly* NP*-complete.*

4 Results: Polynomial Time Approximation Algorithms

In this section, we focus on the polynomial-time approximability of our problems. For all the problems in this paper, we study two natural optimization versions: (i) minimizing the weight of the solution given a target value as input and (ii) maximizing the value of the solution given knapsack size as input. We first consider minimizing the weight of the solution.

A natural integer linear programming formulation of Vertex Cover Knapsack is the following.

minimize $\sum_{u \in \mathcal{V}} w(u)x_u$

Subject to:

$$\begin{aligned} & x_u + x_v \geqslant 1, \forall (u,v) \in \mathcal{E} \\ & \sum_{u \in \mathcal{V}} \alpha(u) x_u \geqslant p \\ & x_u \in \{0,1\}, \forall u \in \mathcal{V} \end{aligned} \tag{1}$$

We replace the constraints $x_u \in \{0,1\}$, with $x_u \geqslant 0$, $\forall u \in \mathcal{V}$ to obtain linear programming (abbreviated as LP) relaxation of the integer linear program (abbreviated as ILP).

▶ **Observation 14.** *The relaxed LP of the ILP 1 has an unbounded integrality gap. To see this, consider an edgeless graph on two vertices v_1 and v_2. Let $w(v_1) = 0$, $w(v_2) = 1$, $\alpha(v_1) = p - 1$ and $\alpha(v_2) = p$. The optimal solution to ILP sets $x_{v_1} = 0, x_{v_2} = 1$, for a total weight of 1. However, the optimal solution to the relaxed LP sets $x_{v_1} = 1, x_{v_2} = 1/p$ and has a total weight of $1/p$. Thus, in this case, the integrality gap is at least $\frac{1}{1/p} = p$.*

To tackle Observation 14, we strengthen the inequality $\sum_{u \in \mathcal{V}} \alpha(u)x_u \geqslant p$. This allows us to obtain an f approximation algorithm even for the more general Set Cover Knapsack problem that we present now. In particular, in addition to having a constraint for every element of the universe, we have a constraint for every $\mathcal{A} \subseteq \mathcal{F}$ of sets such that $\alpha(A) = \sum_{i \in \mathcal{A}} \alpha(i) < p$ where p is the target value given as input. We define the residual value $p_{\mathcal{A}} = p - \alpha(\mathcal{A})$. Given the set $\mathcal{A}$, we simplify the problem on the sets $\mathcal{F} - \mathcal{A}$, where the target value is now $p_{\mathcal{A}}$. We also reduce the value of each set $S_i \in \mathcal{F} - \mathcal{A}$ to be the minimum of its own value and $p_{\mathcal{A}}$, i.e., let $\alpha^{\mathcal{A}}(i) = \min(\alpha(i), p_{\mathcal{A}})$. We can now give the following Integer linear programming formulation of the problem:

minimize $\sum_{i \in [m]} w(i)x_i$

Subject to:

$$\sum_{i : e_j \in \mathcal{S}_i} x_i \geqslant 1, \forall e_j \in \mathcal{U}$$

$$\sum_{i \in \mathcal{F} - \mathcal{A}} \alpha^{\mathcal{A}}(i)x_i \geqslant p_{\mathcal{A}}, \forall \mathcal{A} \subseteq \mathcal{F}$$

$$x_i \in \{0, 1\}, \forall i \in [m]$$

We replace the constraints $x_i \in \{0, 1\}$, with $x_i \geqslant 0$ to obtain the LP relaxation of the ILP. The dual of the LP relaxation is :

maximize $\sum_{\mathcal{A} : \mathcal{A} \subseteq \mathcal{F}} p_{\mathcal{A}} y_{\mathcal{A}} + \sum_{j \in [n]} y_j$

Subject to:

$$\sum_{j : e_j \in \mathcal{S}_i} y_j \leqslant w(i), \forall \mathcal{S}_i \in \mathcal{F}$$

$$\sum_{\mathcal{A} \subseteq \mathcal{F} : i \notin \mathcal{A}} \alpha^{\mathcal{A}}(i) y_{\mathcal{A}} \leqslant w(i), \forall i \in \mathcal{F}$$

$$y_{\mathcal{A}} \geqslant 0, \forall \mathcal{A} \subset \mathcal{F}$$

In our primal-dual algorithm, we begin with dual feasible solution $y = 0$ and partial solution $\mathcal{A} = \emptyset$. We pick one set in every iteration until the value of the set $\mathcal{A}$ of sets becomes at least the target value p. We increase the dual variable $y_{\mathcal{A}}$ in every iteration until the dual constraint for some set $i \in \mathcal{F} - \mathcal{A}$ becomes tight. We then pick that set in our solution and continue. After this loop terminates, the value of the set $\mathcal{A}$ of sets is at least the target value p. At that point, if $\mathcal{A}$ is a set cover, then we output $\mathcal{A}$. Otherwise, till there exists an element e_j of the universe that is not covered by $\mathcal{A}$, we increase the dual variable y_j until the dual constraint for some set ℓ with $e_j \in S_l$ becomes tight. We then include S_ℓ in $\mathcal{A}$ and continue. We present the pseudocode of our algorithm in Algorithm 1.

▶ **Theorem 15.** *Algorithm 1 is an f-approximation algorithm for the* Set Cover Knapsack *problem for minimizing the weight of the solution, where f is the maximum number of sets in the family where any element belongs.*

Algorithm 1 Primal-dual f-approximation algorithm for Set Cover Knapsack.

$y \leftarrow 0, \mathcal{A} \leftarrow \emptyset$
while $\alpha(\mathcal{A}) < p$ **do**
  Increase $y_{\mathcal{A}}$ until for some $i \in \mathcal{F} - \mathcal{A}, \sum_{\mathcal{B} \subseteq \mathcal{F}: i \notin \mathcal{B}} \alpha^{\mathcal{B}}(i) y_{\mathcal{B}} = w(i)$
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{i\}$
end while
$\mathcal{X} \leftarrow \mathcal{A}, \mathcal{A}' \leftarrow \mathcal{A}$
while $\exists e_j \notin \bigcup_{i \in \mathcal{A}'} S_i$ **do**
  Increase y_j until there is some t with $e_j \in S_t$ such that $\sum_{j: e_j \in S_t} y_j = w(t)$
  $\mathcal{A}' \leftarrow \mathcal{A} \cup \{t\}$
end while
return $\mathcal{A}'$

Proof. Let ALG be the weight of the set cover $\mathcal{A}'$ output by Algorithm 1. Then

$$\begin{aligned} \mathsf{ALG} &= \sum_{i \in \mathcal{A}'} w(i) x_i \\ &= \sum_{i \in \mathcal{X}} w(i) x_i + \sum_{i \in \mathcal{A}' - \mathcal{X}} w(i) x_i \end{aligned}$$

Let OPT be the optimal weight of the Set Cover Knapsack instance, set i picked in the i-th iteration of the first while loop (which we can assume without loss of generality by renaming the sets), and l the set selected by the algorithm at the last iteration of the first while loop. Since the first while loop terminates when $\alpha(\mathcal{A}) \geqslant p$, we know that $\alpha(\mathcal{X}) \geqslant p$; since set l was added to $\mathcal{X}$, it must be the case that before l was added, the total value of $\mathcal{A}$ was less than p, so that $\alpha(\mathcal{X} - \{l\}) < p$. For $i \in [l]$, we define $\mathcal{A}_i = [i]$ as the set of sets picked in the first i iterations of the first while loop and $\mathcal{C} = \{\mathcal{A}_i : i \in [l]\}$. We observe that a dual variable $y_{\mathcal{B}}$ is non-zero only if $\mathcal{B} \in \mathcal{C}$. Since we pick only tight sets, we have

$$\sum_{i \in \mathcal{X}} w(i) = \sum_{i \in \mathcal{X}} \sum_{\mathcal{B} \subseteq \mathcal{F}: i \notin \mathcal{B}} \alpha^{\mathcal{B}}(i) y_{\mathcal{B}} = \sum_{i \in \mathcal{X}} \sum_{\mathcal{B} \in \mathcal{C}: i \notin \mathcal{B}} \alpha^{\mathcal{B}}(i) y_{\mathcal{B}}.$$

Reversing the double sum, we have

$$\sum_{i \in \mathcal{X}} \sum_{\mathcal{B} \in \mathcal{C}: i \notin \mathcal{B}} \alpha^{\mathcal{B}}(i) y_{\mathcal{B}} = \sum_{\mathcal{B} \in \mathcal{C}} y_{\mathcal{B}} \sum_{i \in \mathcal{X} - \mathcal{B}} \alpha^{\mathcal{B}}(i).$$

Note that in any iteration of the algorithm except the last one, adding the next set i to the current sets in $\mathcal{A}$ did not cause the value of the knapsack to become at least p; that is, $\alpha(i) < p - \alpha(\mathcal{A}) = p_{\mathcal{A}}$ at that point in the algorithm. Thus, for all sets $i \in \mathcal{A}$ except l, $\alpha^{\mathcal{A}}(i) = \min(\alpha(i), p_{\mathcal{A}}) = \alpha(i)$, for the point in the algorithm at which $\mathcal{A}$ was the current set of sets. Thus, we can rewrite

$$\sum_{i \in \mathcal{X} - \mathcal{A}} \alpha^{\mathcal{A}}(i) = \alpha^{\mathcal{A}}(l) + \sum_{i \in \mathcal{X} - \mathcal{A}: i \neq l} \alpha^{\mathcal{A}}(i) = \alpha^{\mathcal{A}}(l) + \alpha(\mathcal{X} - \{l\}) - \alpha(\mathcal{A}).$$

Note that $\alpha^{\mathcal{A}}(l) \leqslant p_{\mathcal{A}}$ by definition, and as argued at the beginning of the proof $\alpha(\mathcal{X} - \{l\}) < p$ so that $\alpha(\mathcal{X} - \{l\}) - \alpha(\mathcal{A}) < p - \alpha(\mathcal{A})) = p_{\mathcal{A}}$; thus, we have that

$$\alpha^{\mathcal{A}}(l) + \alpha(\mathcal{X} - \{l\}) - \alpha(\mathcal{A}) < 2p_{\mathcal{A}}$$

which is the same as saying

$$\sum_{i \in \mathcal{X} - \mathcal{B}} \alpha^{\mathcal{B}}(i) < 2p_{\mathcal{B}} \text{ for every } \mathcal{B} \in \mathcal{C}.$$

Therefore,

$$\sum_{i \in \mathcal{X}} w(i) = \sum_{\mathcal{B} \in \mathcal{C}} y_{\mathcal{B}} \sum_{i \in \mathcal{X} - \mathcal{B}} \alpha^{\mathcal{B}}(i) < 2 \sum_{\mathcal{B}: \mathcal{B} \in \mathcal{C}} p_{\mathcal{B}} y_{\mathcal{B}} = 2 \sum_{\mathcal{B} \subseteq \mathcal{F}: i \notin \mathcal{B}} p_{\mathcal{B}} y_{\mathcal{B}}$$

where the last equality follows from the fact that $y_{\mathcal{B}} = 0$ if $\mathcal{B} \notin \mathcal{C}$.

Our algorithm picks sets in $\mathcal{A}\prime \setminus \mathcal{X}$ in the second while loop if the set of sets picked in the first while loop does not form a set cover. We now upper bound $\sum_{i \in \mathcal{A}' \setminus \mathcal{X}} w(i)$ as follows.

$$\sum_{i \in \mathcal{A}' \setminus \mathcal{X}} w(i) = \sum_{i \in \mathcal{A}' \setminus \mathcal{X}} \sum_{j \in [n]: e_j \in S_i} y_j = \sum_{j \in [n]} |\{i \in \mathcal{A}' \setminus \mathcal{X} : e_j \in S_i\}| y_j \leqslant f \sum_{j \in [n]} y_j$$

The first equality follows from the fact that only tight sets are picked. We now bound ALG.

$$\begin{aligned}
\mathsf{ALG} &= \sum_{i \in \mathcal{A}'} w(i) x_i \\
&= \sum_{i \in \mathcal{X}} w(i) x_i + \sum_{i \in \mathcal{A}' - \mathcal{X}} w(i) x_i \\
&\leqslant 2 \sum_{A: A \subseteq I} p_{\mathcal{A}} y_{\mathcal{A}} + f \sum_{j \in [n]} y_j \\
&\leqslant f \left(\sum_{A: A \subseteq I} p_{\mathcal{A}} y_{\mathcal{A}} + \sum_{j \in [n]} y_j \right) \\
&= f\mathsf{OPT}
\end{aligned}$$

◀

We note that our algorithm is a combinatorial algorithm based on the primal-dual framework – in particular, we use LPs only to design and analyze our algorithm. We do not need to solve any LP. We obtain approximation algorithms for the Vertex Cover Knapsack and d-Hitting Set Knapsack problems as corollaries of Theorem 15 by reducing these problems to Set Cover Knapsack.

▶ **Corollary 16** (⋆). *There exists a* ***d****-approximation algorithm for* d-Hitting Set Knapsack *for minimizing the weight of the solution. The algorithm is combinatorial in nature and based on the primal-dual method.*

▶ **Corollary 17** (⋆). *There exists a* 2*-approximation algorithm for* Vertex Cover Knapsack *for minimizing the weight of the solution. The algorithm is combinatorial in nature and based on the primal-dual method.*

We next present a H_d-approximation algorithm for Set Cover Knapsack where d is the maximum cardinality of any set in the input instance and $H_d = \sum_{i=1}^{d} \frac{1}{i}$ is the d-th harmonic number. The idea is to run the first while loop of Algorithm 1, and then, if the selected sets do not cover the universe, then, instead of the second while loop of Algorithm 1, we pick sets following the standard greedy algorithm for minimum weight set cover. We show that the algorithm achieves an approximation factor of $\max(2, H_d)$ by analyzing it using the *dual fitting technique*.

Algorithm 2 Max(2, H_d)-approximation algorithm for Set Cover Knapsack.

$y \leftarrow 0, \mathcal{A} \leftarrow \emptyset$
while $\alpha(\mathcal{A}) < p$ **do**
  Increase $y_{\mathcal{A}}$ until for some $i \in \mathcal{F} - \mathcal{A}, \sum_{\mathcal{B} \subseteq \mathcal{F}: i \notin \mathcal{B}} \alpha^{\mathcal{B}}(i) y_{\mathcal{B}} = w(i)$
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{i\}$
end while
$\mathcal{X} \leftarrow \mathcal{A}, \mathcal{U}' \leftarrow \mathcal{U} - \bigcup_{i \in \mathcal{X}} \mathcal{S}_i, \mathcal{F}' \leftarrow \mathcal{F} - \mathcal{X}, I \leftarrow \emptyset, \hat{S}_i \leftarrow S_i$ for all $i \in \mathcal{F}'$
while I is not a set cover for $\mathcal{U}'$ **do**
  $l \leftarrow \arg\min_{i: \hat{S}_i \neq \emptyset} \frac{w(i)}{|\hat{S}_i|}$
  $I \leftarrow I \cup \{l\}$
  $\hat{S}_i \leftarrow \hat{S}_i - S_l$ for all $i \in \mathcal{F}'$
end while
return $\mathcal{X} \cup \mathcal{I}$

▶ **Theorem 18.** *Algorithm 2 is a max(2, H_d)-approximation algorithm for the* Set Cover Knapsack *problem for minimizing the weight of the solution, where d is the maximum cardinality of any set in the input.*

Proof. We follow the same notation defined in Algorithm 2 in this proof. Since the first part of Algorithm 2 is the same as the first part of Algorithm 1, from the proof of Theorem 15, we have

$$\sum_{i \in \mathcal{X}} w(i) < 2 \sum_{\mathcal{B} \subseteq \mathcal{F}: i \notin \mathcal{B}} p_{\mathcal{B}} y_{\mathcal{B}}.$$

To bound the sum of weights of the sets in $\mathcal{I}$, we use the dual fitting technique. In particular, we will first construct an assignment of dual variables $(y_j)_{j \in [n]}$ with $\sum_{i \in \mathcal{I}} w_i = \sum_{j=1}^{n} y_j$. However, $(y_j)_{j \in [n]}$ may not satisfy the dual constraints involving those variables. However, and then show that $y'_j = \frac{1}{H_d} y_j, j \in [n]$ satisfies all dual constraints involving those variables. We concretize this idea below.

Whenever Algorithm 2 includes a set $\hat{\mathcal{S}}_i$ in $\mathcal{I}$, we define $y_j = \frac{w(i)}{|\hat{\mathcal{S}}_i|}$ for each $j \in \hat{\mathcal{S}}_i$. Since each $j \in \hat{\mathcal{S}}_i$ is uncovered in iteration when Algorithm 2 picks the set $\hat{\mathcal{S}}_i$, and is then covered for the remaining iterations of the algorithm (because we added subset $\mathcal{S}_i$ to $\mathcal{I}$), the dual variable y_j is set to a value exactly once. Furthermore, we see that

$$w(i) = \sum_{i: j \in \hat{\mathcal{S}}_i} y_j, \forall i \in \mathcal{I}$$

since the weight $w(i)$ of the set i is distributed among $y_j, j \in \hat{\mathcal{S}}_i$. Hence, we have,

$$\sum_{j \in \mathcal{I}} w(i) = \sum_{i=1}^{n} y_j.$$

We claim that $y'_j = \frac{1}{H_d} y_j$ for all $j \in [n]$ satisfies the dual constraints involving these variables. For that, we need to show that for each subset $\mathcal{S}_i, i \in [m]$,

$$\sum_{i: j \in \mathcal{S}_i} y'_j \leqslant w(i).$$

Pick an arbitrary subset $\mathcal{S}_i$ and an arbitrary iteration k of the second while loop of Algorithm 2. Let ℓ be the number of iterations that the second while loop of Algorithm 2 makes and a_k the number of elements in this subset that is still uncovered at the beginning of the k-th

iteration, so that $a_1 = |\mathcal{S}_i|$, and $a_{\ell+1} = 0$. Let A_k be the set of uncovered elements of $\mathcal{S}_i$ covered in the k-th iteration, so that $|A_k| = a_k - a_{k+1}$. If subset $\mathcal{S}_q$ is chosen in the k-th iteration, then for each element $j \in A_k$ covered in the k-th iteration, we have

$$y'_j = \frac{w_q}{H_d|\hat{\mathcal{S}}_q|} \leqslant \frac{w(i)}{H_d a_k},$$

where $\hat{\mathcal{S}}_q$ is the set of uncovered elements of $\mathcal{S}_q$ at the beginning of the k-th iteration. The inequality follows because if $\mathcal{S}_q$ is chosen in the k-th iteration, it must minimize the ratio of its weight to the number of uncovered elements it contains. Thus,

$$\begin{aligned}
\sum_{i:e_j\in\mathcal{S}_i} y'_j &= \sum_{k=1}^{l} \sum_{j\in[n]:j\in A_k} y'_j \\
&\leqslant \sum_{k=1}^{l} (a_k - a_{k+1}) \frac{w(i)}{H_d a_k} \\
&\leqslant \frac{w(i)}{H_d} \sum_{k=1}^{l} \frac{a_k - a_{k+1}}{a_k} \\
&\leqslant \frac{w(i)}{H_d} \sum_{k=1}^{l} \left(\frac{1}{a_k} + \frac{1}{a_k - 1} + \cdots + \frac{1}{a_{k+1} + 1} \right) \\
&\leqslant \frac{w(i)}{H_d} \sum_{i=1}^{|\mathcal{S}_i|} \frac{1}{i} \\
&= \frac{w(i)}{H_d} H_{|\mathcal{S}_i|} \\
&\leqslant w(i),
\end{aligned}$$

where the final inequality follows because $|\mathcal{S}_i| \leqslant d$. Hence, $((y_{\mathcal{B}})_{\mathcal{B}\in\mathcal{F}}, (y'_j)_{j\in[n]})$ is a dual feasible solution. We now bound ALG as follows.

$$\begin{aligned}
\mathsf{ALG} &= \sum_{i\in\mathcal{X}} w(i)x_i + \sum_{i\in\mathcal{I}} w(i)x_i \\
&\leqslant 2 \sum_{A:A\subseteq\mathcal{F}} p_{\mathcal{A}} y_{\mathcal{A}} + H_d \sum_{j\in[n]} y_j \\
&= \max(2, H_d) \left(\sum_{A:A\subseteq\mathcal{F}} p_{\mathcal{A}} y_{\mathcal{A}} + \sum_{j\in[n]} y_j \right) \\
&= \max(2, H_d) \cdot \mathsf{OPT}
\end{aligned}$$

◀

The approximation guarantees of Theorems 15 and 18 are the best possible approximation guarantees, up to any additive constant $\varepsilon > 0$, that any polynomial time algorithm hopes to achieve, assuming standard complexity-theoretic assumptions.

▶ **Theorem 19** ($\star$). *Let $\varepsilon > 0$ be any constant. Then we have the following:*

1. *There is no polynomial-time $(1-\varepsilon)\ln n$ factor approximation algorithm for* Set Cover Knapsack *unless every problem in* NP *admits a quasi-polynomial time algorithm.*
2. *Assuming Unique Games Conjecture (UGC), there is no polynomial-time $(1-f)\ln n$ factor approximation algorithm for* Set Cover Knapsack.

3. *Assuming Unique Games Conjecture (UGC), there is no polynomial-time* $(1-d)\ln n$ *factor approximation algorithm for* d-Hitting Set Knapsack.

We next focus on maximizing the value of the solution given a knapsack size as input. Surprisingly, for all the problems studied in this paper, we show that there is no ρ-approximation algorithm for any of our problems for any $\rho > 1$.

▶ **Theorem 20** (⋆). *For any* $\rho > 1$*, there does not exist a* ρ*-approximation algorithm for maximizing the value of the solution given the size of the knapsack for* Set Cover Knapsack*,* d-Hitting Set Knapsack*,* Vertex Cover Knapsack*,* Minimal Vertex Cover Knapsack*,* Minimum Vertex Cover Knapsack*, and* k-Vertex Cover Knapsack *unless* $\mathsf{P} = \mathsf{NP}$*.*

The inapproximability barriers of Theorems 19 and 20 can be overcome using the framework of FPT-approximation. In particular, we will show FPT $(1-\varepsilon)$-approximation algorithms, parameterized by the treewidth of the input graph, for all four variants of vertex cover knapsack for maximizing the value of the solution.

5 Results: Parameterized Complexity

We study the four variants of Vertex Cover Knapsack using the framework of parameterized complexity. For that, we consider the treewidth of the input graph as a parameter. With respect to treewidth, we design algorithms that run in time single exponential in the treewidth times polynomial in n (number of vertices), size s, and target value p of the knapsack. We then use these algorithms to develop a $(1-\varepsilon)$-approximation algorithm for maximizing the value of the solution that runs in time single exponential in the treewidth times polynomial in the number n of vertices, $1/\varepsilon$ and $\sum_{v\in\mathcal{V}} \alpha(v)$.

We know that there exists a $\mathcal{O}\left(2^{\mathrm{tw}} \cdot \mathrm{tw}^{\mathcal{O}(1)} \cdot n\right)$ time algorithm for the Vertex Cover problem [6]. It turns out that it is relatively easy to modify that algorithm to design a similar algorithm Vertex Cover Knapsack, k-Vertex Cover Knapsack, and Minimum Vertex Cover Knapsack.

▶ **Theorem 21** (⋆). *There is an algorithm for* Vertex Cover Knapsack *with running time* $\mathcal{O}\left(2^{tw} \cdot n^{\mathcal{O}(1)} \cdot \min\{s^2, p^2\}\right)$.

It turns out that the main idea of the algorithm of Theorem 21 can be modified to obtain algorithms for Minimum Vertex Cover Knapsack and k-Vertex Cover Knapsack with similar running times.

▶ **Theorem 22** (⋆). *There is an algorithm for* Minimum Vertex Cover Knapsack *with running time* $\mathcal{O}\left(2^{tw} \cdot n^{\mathcal{O}(1)} \cdot \min\{s^2, p^2\}\right)$.

▶ **Theorem 23** (⋆). *There is an algorithm for* k-Vertex Cover Knapsack *with running time* $\mathcal{O}\left(2^{tw} \cdot n^{\mathcal{O}(1)} \cdot \min\{s^2, p^2\}\right)$.

However, the approach of Theorem 21 breaks down for Minimal Vertex Cover Knapsack. This is so because a minimal vertex cover (unlike a vertex cover, a vertex cover of size at most k, and a minimum vertex cover) of a graph may not be a minimal vertex cover of some of its induced subgraphs. For this reason, it is not enough to keep track of all minimal vertex covers of the subgraphs rooted at some tree node intersecting the bag at certain subsets. Intuitively speaking, we tackle this problem by adding another subset of vertices in the "indices" of the DP table that will be part of some minimal vertex cover of some other induced subgraph.

▶ **Theorem 24.** *There is an algorithm for* Minimal Vertex Cover Knapsack *with running time* $\mathcal{O}\left(16^{tw} \cdot n^{\mathcal{O}(1)} \cdot \min\{s^2, p^2\}\right)$.

Proof. Let $(G = (V, E), (w(u))_{u \in V}, (\alpha(u))_{u \in V}, s, p)$ be an input instance of Minimal Vertex Cover Knapsack and $(\mathbb{T} = (V_{\mathbb{T}}, E_{\mathbb{T}}), \mathcal{X})$ a nice tree decomposition rotted at node r of treewidth $\mathrm{tw}(G)$.

We define a function $\ell : V_{\mathbb{T}} \to \mathbb{N}$. For a vertex $t \in V_{\mathbb{T}}$, $\ell(t) = \mathsf{dist}_{\mathbb{T}}(\mathsf{t}, \mathsf{r})$, where r is the root. Note that this implies that $\ell(r) = 0$. Let us assume that the values that ℓ take over the nodes of $\mathbb{T}$ are between 0 and L. For a node $t \in V_{\mathbb{T}}$, we denote the set of vertices in the bags in the subtree rooted at t by V_t and $G_t = G[V_t]$. We now describe a dynamic programming algorithm over $(\mathbb{T}, \mathcal{X})$ for Minimal Vertex Cover Knapsack.

States. We maintain a DP table D where a state has the following components:

1. t represents a node in $V_{\mathbb{T}}$.
2. V_1, V_2 are subsets of the bag X_t, not necessarily disjoint.
3. V_1 represents the intersection of X_t with a minimal vertex cover of the subgraph $G_t[(V_t \setminus V_2) \cup V_1]$.

Interpretation of States. For each node $t \in \mathbb{T}, V_1, V_2 \subseteq X_t$ and "undominated" minimal vertex cover S of the induced graph $G_t[(V_t \setminus V_2) \cup V_1]$ such that $S \cap X_t = V_1$, we store $(w(S), \alpha(S))$ in the list $D[t, V_1, V_2]$. We say a minimal vertex cover $S_1 \subseteq (V_t \setminus V_2) \cup V_1$ dominates another minimal vertex cover $S_2 \subseteq (V_t \setminus V_2) \cup V_1$ if $w(S_1) \leqslant w(S_2)$ and $\alpha(S_1) \geqslant \alpha(S_2)$ with at least one inequality being strict. We say a minimal vertex cover of $G_t[(V_t \setminus V_2) \cup V_1]$ *undominated* if no other minimal vertex cover of $G_t[(V_t \setminus V_2) \cup V_1]$ dominates it.

For each state $D[t, V_1, V_2]$, we initialize $D[t, V_1, V_2]$ to the list $\{(0, 0)\}$.

Dynamic Programming on D. We first update the table for states with nodes $t \in V_{\mathbb{T}}$ such that $\ell(t) = L$. When all such states are updated, then we update states where the level of node t is $L - 1$, and so on, till we finally update states with r as the node – note that $\ell(r) = 0$. For a particular j, $0 \leqslant j < L$ and a state $[t, V_1, V_2]$ such that $\ell(t) = j$, we can assume that $D[t, V_1, V_2]$ have been evaluated for all t', such that $\ell(t') > j$ and all subsets V_1' and V_2' of $X_{t'}$. Now we consider several cases by which $D[t, V_1, V_2]$ is updated based on the nature of t in $\mathbb{T}$:

1. Suppose t is a leaf node with $X_t = \{v\}$. Then $D[t, v, \emptyset] = (w(v), \alpha(v))$, or $D[t, \emptyset, v] = (0, 0)$ and $D[t, \emptyset, \emptyset]$ stores the pair $(0, 0)$.
2. Suppose t is an introduce node. Then it has an only child t' where $X_{t'} \cup \{u\} = X_t$. Then for all $\mathbb{S} \subseteq X_t$: If $\mathbb{S}$ is not a vertex cover of $G[X_t]$, we set $D[t, V_1, V_2] = (\infty, \infty)$. Otherwise, we have three cases:
   a. Case 1: If $u \notin V_1 \cup V_2$, then we copy each pair (w, α) from $D[t', V_1, V_2]$
   b. Case 2: If $u \in V_1 \setminus V_2$, then
      i. we check if $N(u) \setminus V_1 \neq \emptyset$, then
         A. for each pair (w, α) in $D[t', V_1 \setminus \{u\}, V_2]$, if $w + w(u) \leqslant s$, then we put $(w + w(u), \alpha + \alpha(u))$ in $D[t', V_1 \setminus \{u\}, V_2]$ to $D[t, V_1, V_2]$.
         B. for each pair (w, α) in $D[t', V_1 \setminus \{u\}, V_2]$, if $w + w(u) > s$, we put (w, α) to $D[t, V_1, V_2]$.
      ii. Otherwise we store (∞, ∞).
   c. Case 3: If $u \in V_2$, then we set $D[t, V_1, V_2] = D[t', V_1, V_2 \setminus \{u\}]$.
3. Suppose t is a forget vertex node. Then it has an only child t' where $X_t \cup \{u\} = X_{t'}$. We copy all the pairs from $D[t', V_1 \cup \{u\}, V_2]$, $D[t', V_1, V_2 \cup \{u\}]$ and $D[t', V_1, V_2]$ to $D[t, V_1, V_2]$ and remove all dominated pairs.

4. Suppose t is a join node. Then it has two children t_1, t_2 such that $X_t = X_{t_1} = X_{t_2}$. Let $(w(V_1 \cap V_2), \alpha(V_1 \cap V_2))$ be the total weight and value of the vertices in $V_1 \cap V_2$. Then for all $W_1, W_2 \subseteq V_1 \subseteq X_t$, consider a pair (w_1, α_1) in $D[t_1, W_1, W_2 \cup V_2]$ and a pair (w_2, α_2) in $D[t_2, W_2, W_1 \cup V_2]$. Suppose $w_1 + w_2 - w(V_1 \cap V_2) \leqslant s$. Then we add $(w_1 + w_2 - w(V_1 \cap V_2), \alpha_1 + \alpha_2 - \alpha(V_1 \cap V_2))$ to $D[t, V_1, V_2]$.

Finally, in the last step of updating $D[t, V_1, V_2]$, we go through the list saved in $D[t, V_1, V_2]$ and only keep undominated pairs.

Correctness of the algorithm. Recall that we are looking for a solution $\mathcal{U}$ that contains the fixed vertex v that belongs to all bags of the tree decomposition. In each state we maintain the invariant $V_1, V_2 \subseteq X_t$ such that $V_1 = X_t \cap$ minimal vertex cover knapsack of $G_t \setminus$ edges incident on $V_2 \setminus V_1$. First, we show that a pair (w, α) belonging to $D[t, V_1, V_2]$ for a node $t \in V_{\mathbb{T}}$ and a subset $\mathbb{S}$ of X_t corresponds to a minimal vertex cover knapsack H in G_t. Recall that $X_r = \{v\}$. Thus, this implies that a pair (w, α) belonging to $D[r, V_1 = \{v\}, V_2 = \emptyset)]$ corresponds to a minimal vertex cover knapsack of G. Moreover, the output is a pair that is feasible and with the highest value.

In order to show that a pair (w, α) belonging to $D[t, V_1, V_2]$ for a node $t \in V_{\mathbb{T}}$ and a subset V_1 of X_t corresponds to a minimal vertex cover knapsack of $G_t \setminus$ edges incident on $V_2 \setminus V_1$, we need to consider the cases of what t can be:

1. **Leaf node**: Recall that in our modified nice tree decomposition we have added a vertex v to all the bags. Suppose a leaf node t contains a single vertex v, $D[t, v, \emptyset] = (w(v), \alpha(v))$, $D[t, \emptyset, v] = (0, 0)$ and $D[L, \emptyset, \emptyset]$ stores the pair $(0, 0)$. This is true in particular when $j = L$, the base case. From now we can assume that for a node t with $\ell(t) = j < L$ and all subsets $V_1, V_2 \subseteq X_t$, $D[t', V_1'', V_2'']$ entries are correct and correspond to minimal vertex cover in $G_{t'} \setminus$ edges incident on $V_2'' \setminus V_1''$. when $\ell(t') > j$.
2. **Introduce node**: When t is an introduce node, there is a child t'. We are introducing a vertex u and the edges associated with it in G_t. Since $\ell(t') > \ell(t)$, by induction hypothesis all entries in $D[t', V_1'' = V_1 \setminus \{u\}, V_2'' = V_2 \setminus \{u\}]$, $D[t', V_1'' = V_1 \setminus \{u\}, V_2'' = V_2]$, and $D[t', V_1'' = V_1, V_2'' = V_2 \setminus \{u\}]$, $\forall\ V_1'', V_2'' \subseteq X_{t'}$ are already computed and feasible. We update pairs in $D[t, V_1, V_2]$ from $D[t', V_1 \setminus \{u\}, V_2]$ or $D[t', V_1, V_2 \setminus \{u\}]$ or $D[t', V_1 \setminus \{u\}, V_2 \setminus \{u\}]$ such that either u is considered as part of a minimal solution in $G_t \setminus$ edges incident on $V_2 \setminus V_1$ or not.
3. **Forget node**: When t is a forget node, there is a child t'. We are forgetting a vertex u and the edges associated with it in G_t. Since $\ell(t') > \ell(t)$, by induction hypothesis all entries in $D[t', V_1'' = V_1 \cup \{u\}, V_2'' = V_2]$, $D[t', V_1'' = V_1, V_2'' = V_2 \cup \{u\}]$, and $D[t', V_1'' = V_1, V_2'' = V_2]$, $\forall\ V_1'', V_2'' \subseteq X_{t'}$ are already computed and feasible. We copy each undominated (w, α) pair stored in $D[t', V_1 \cup \{u\}, V_2]$, $D[t', V_1, V_2 \cup \{u\}]$ and $D[t', V_1, V_2]$ to $D[t, V_1, V_2]$.
4. **Join node**: When t is a join node, there are two children t_1 and t_2 of t, such that $X_t = X_{t_1} = X_{t_2}$. For all subsets $V_1 \subseteq X_t$ we partition V_1 into two subsets W_1 and W_2 (not necessarily disjoint) such that W_1 is the intersection of X_{t_1} with minimal solution in the graph $G_{t_1} \setminus$ edges incident on $(W_2 \cup V_2) \setminus W_1$. Similarly, W_2 is the intersection of X_{t_2} with minimal solution in the graph $G_{t_2} \setminus$ edges incident on $(W_1 \cup V_2) \setminus W_2$. By the induction hypothesis, the computed entries in $D[t_1, W_1, W_2 \cup V_2]$ and $D[t_2, W_2, W_1 \cup V_2]$ where $W_1 \cup W_2 = V_1$ are correct and store the non redundant minimal vertex cover for the subgraph G_{t_1} in W_1 and similarly, W_2 for G_{t_2}. Now we add $(w_1 + w_2 - w(V_1 \cap V_2), \alpha_1 + \alpha_2 - \alpha\{V_1 \cap V_2)))$ to $D[t, V_1, V_2]$.

What remains to be shown is that an undominated feasible solution $\mathcal{U}$ of Minimal Vertex Cover Knapsack in G is contained in $D[r, \{v\}, \emptyset]$. Let w be the weight of $\mathcal{U}$ and α be the value. Recall that $v \in \mathcal{U}$. For each t, we consider the subgraph G_t and observe how

the minimal solution $\mathcal{S}'$ interacts with G_t. Let $\hat{V}_1, \hat{V}_2, \ldots, \hat{V}_m$ be components of $G_t \cap \mathcal{U}$ and let for each $1 \leqslant i \leqslant m$, $\mathcal{S}_i = X_t \cap \hat{V}_i$. Also, let $\hat{V}_0 = X_t \setminus \mathcal{U}$. Consider $\mathcal{S} = (\hat{V}_0, \hat{V}_1, \hat{V}_2, \ldots, \hat{V}_m)$. For each $\mathcal{S}_i$, we define subsets V_1 and V_2 such that $V_1, V_2 \subseteq \mathcal{S}_i$, and $V_1 \cup V_2 = \mathcal{S}_i$, $\forall i \in [m]$. The algorithm updates in $D[t, V_1, V_2]$ the pair (w', α') for the subsolution ($G_t \setminus$ edges incident on V_2) $\cap \mathcal{U}$. Therefore, $D[r, \{v\}, \emptyset]$ contains the pair (w, α). Thus, we are done.

Running Time. There are n choices for the fixed vertex v. Upon fixing v and adding it to each bag of $(\mathbb{T}, \mathcal{X})$, we consider the total possible number of states. Observe that the number of subproblems is small: for every node t, we have only $2^{|X_t|}$ choices for V_1 and V_2. Hence, the number of entries of the DP table is $\mathcal{O}(4^{\mathrm{tw}} \cdot n)$. For each state, since we are keeping only undominated pairs, for each weight w there can be at most one pair with w as the first coordinate; similarly, for each value α there can be at most one pair with α as the second coordinate. Thus, the number of undominated pairs in each $D[t, V_1, V_2]$ is at most $\min\{s, p\}$ that can be maintained in time $\min\{s^2, p^2\}$. Updating the entries of the join nodes has the highest time complexity among all tree nodes, which is $\mathcal{O}\left(4^{\mathrm{tw}} \cdot n^{\mathcal{O}(1)}\right)$. Hence, the overall running time of our algorithm is $\mathcal{O}\left(16^{\mathrm{tw}} \cdot n^{\mathcal{O}(1)} \cdot \min\{s^2, p^2\}\right)$. ◀

We now design a fully FPT-time approximation scheme for the Minimal Vertex Cover Knapsack problem by rounding the values of the items so that $\alpha(\mathcal{V})$ is indeed a polynomial in n. The idea is to scale down the value of every vertex of the input instance so that the sum of values of the vertices that can be in the solution is polynomially bounded by input length and solve the scaled-down instance using the algorithm in Theorem 24. We usually scale down the values by dividing by $(\varepsilon \alpha_{\max})/n$. However, this approach does not work for our problems since $\alpha_{\max}$ is a lower bound on the optimal value for classical knapsack but not necessarily for our vertex cover knapsack variants. We tackle this issue by iteratively guessing upper and lower bounds of OPT thereby incurring an extra factor of $\mathrm{poly}\left(\sum_{v \in \mathcal{V}} \alpha(v)\right)$.

▶ **Theorem 25** (⋆). *For every $\varepsilon > 0$, there is an $(1-\varepsilon)$-factor approximation algorithm for* Minimal Vertex Cover Knapsack *for optimizing the value of the solution and running in time $\mathcal{O}\left(16^{tw} \cdot poly\left(n, 1/\varepsilon, \log\left(\sum_{v \in \mathcal{V}} \alpha(v)\right)\right)\right)$ where tw is the treewidth of the input graph.*

We obtain similar results for the other three variants of vertex cover knapsack for optimizing the value of the solution.

▶ **Corollary 26** (⋆). *For every $\varepsilon > 0$, there are $(1-\varepsilon)$-factor approximation algorithms for* Vertex Cover Knapsack, Minimum Vertex Cover Knapsack, *and* k-Vertex Cover Knapsack *for optimizing the value of the solution and running in time $\mathcal{O}\left(2^{tw} \cdot poly\left(n, 1/\varepsilon, \log\left(\sum_{v \in \mathcal{V}} \alpha(v)\right)\right)\right)$.*

It turns out that we can use a similar idea as in Theorem 25 and Corollary 26 to design an FPT time $(1+\varepsilon)$-approximation algorithm, parameterized by treewidth, for all the variants of vertex cover knapsack for minimizing the weight of the solution for every $\varepsilon > 0$.

▶ **Theorem 27** (⋆). *For every $\varepsilon > 0$, we have the following.*

1. *There is a $(1+\varepsilon)$-factor approximation algorithm for* Minimal Vertex Cover Knapsack *for optimizing the weight of the solution and running in time $\mathcal{O}\left(16^{tw} \cdot poly\left(n, 1/\varepsilon, \log\left(\sum_{v \in \mathcal{V}} w(v)\right)\right)\right)$ where tw is the treewidth of the input graph.*
2. *There are $(1+\varepsilon)$-factor approximation algorithms for* Vertex Cover Knapsack, Minimum Vertex Cover Knapsack, *and* k-Vertex Cover Knapsack *for optimizing the weight of the solution and running in time $\mathcal{O}\left(2^{tw} \cdot poly\left(n, 1/\varepsilon, \log\left(\sum_{v \in \mathcal{V}} w(v)\right)\right)\right)$.*

6 Conclusion

We have studied the classical Knapsack problem with the graph theoretic constraints, namely vertex cover and its interesting variants like Minimum Vertex Cover Knapsack, Minimal Vertex Cover Knapsack, and k-Vertex Cover Knapsack. We further generalize this to hypergraphs and study Set Cover Knapsack and d-Hitting Set Knapsack. We have presented approximation algorithms for minimizing the size of the solution and proved that the approximation factors are the best possible that one hopes to achieve in polynomial time under standard complexity-theoretic assumptions. However, to maximize the value of the solution, we obtain strong inapproximability results. Fortunately, we show that there exist FPT algorithms parameterized by the treewidth of the input graph (for vertex cover variants of knapsack), which can achieve $(1-\varepsilon)$-approximate solution.

References

1 Andrea Bettinelli, Valentina Cacchiani, and Enrico Malaguti. A branch-and-bound algorithm for the knapsack problem with conflict graph. *INFORMS J. Comput.*, 29(3):457–473, 2017. doi:10.1287/ijoc.2016.0742.

2 Flavia Bonomo and Diego de Estrada. On the thinness and proper thinness of a graph. *Discret. Appl. Math.*, 261:78–92, 2019. doi:10.1016/J.DAM.2018.03.072.

3 Nicolas Boria, Federico Della Croce, and Vangelis Th. Paschos. On the max min vertex cover problem. *Discret. Appl. Math.*, 196:62–71, 2015. doi:10.1016/J.DAM.2014.06.001.

4 Nicolas Boria, Federico Della Croce, and Vangelis Th Paschos. On the max min vertex cover problem. *Discrete Applied Mathematics*, 196:62–71, 2015. doi:10.1016/J.DAM.2014.06.001.

5 Stefano Coniglio, Fabio Furini, and Pablo San Segundo. A new combinatorial branch-and-bound algorithm for the knapsack problem with conflicts. *Eur. J. Oper. Res.*, 289(2):435–455, 2021. doi:10.1016/j.ejor.2020.07.023.

6 Marek Cygan, Fedor V Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015. doi:10.1007/978-3-319-21275-3.

7 Peter Damaschke. Parameterized algorithms for double hypergraph dualization with rank limitation and maximum minimal vertex cover. *Discrete Optimization*, 8(1):18–24, 2011. doi:10.1016/J.DISOPT.2010.02.006.

8 Palash Dey, Ashlesha Hota, Sudeshna Kolay, and Sipra Singh. Knapsack with vertex cover, set cover, and hitting set, 2024. arXiv:arXiv:2406.01057.

9 Palash Dey, Sudeshna Kolay, and Sipra Singh. Knapsack: Connectedness, path, and shortest-path. In *Latin American Symposium on Theoretical Informatics*, pages 162–176. Springer, 2024. doi:10.1007/978-3-031-55601-2_11.

10 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

11 Herbert Fleischner, Gert Sabidussi, and Vladimir I. Sarvanov. Maximum independent sets in 3- and 4-regular hamiltonian graphs. *Discret. Math.*, 310(20):2742–2749, 2010. doi:10.1016/j.disc.2010.05.028.

12 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

13 Steffen Goebbels, Frank Gurski, and Dominique Komander. The knapsack problem with special neighbor constraints. *Math. Methods Oper. Res.*, 95(1):1–34, 2022. doi:10.1007/s00186-021-00767-5.

14 Frank Gurski and Carolin Rehs. Solutions for the knapsack problem with conflict and forcing graphs of bounded clique-width. *Math. Methods Oper. Res.*, 89(3):411–432, 2019. doi:10.1007/s00186-019-00664-y.

15 Xin Han, Kazuo Iwama, Rolf Klein, and Andrzej Lingas. Approximating the maximum independent set and minimum vertex coloring on box graphs. In Ming-Yang Kao and Xiang-Yang Li, editors, *Algorithmic Aspects in Information and Management, Third International Conference, AAIM 2007, Portland, OR, USA, June 6-8, 2007, Proceedings*, volume 4508 of *Lecture Notes in Computer Science*, pages 337–345. Springer, 2007. doi:10.1007/978-3-540-72870-2_32.

16 David G. Harris and N. S. Narayanaswamy. A faster algorithm for vertex cover parameterized by solution size. In Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov, editors, *41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024, March 12-14, 2024, Clermont-Ferrand, France*, volume 289 of *LIPIcs*, pages 40:1–40:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.STACS.2024.40.

17 Stephan Held, William J. Cook, and Edward C. Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Math. Program. Comput.*, 4(4):363–381, 2012. doi:10.1007/s12532-012-0042-3.

18 Mhand Hifi and Mustapha Michrafy. A reactive local search-based algorithm for the disjunctively constrained knapsack problem. *Journal of the Operational Research Society*, 57(6):718–726, 2006. doi:10.1057/PALGRAVE.JORS.2602046.

19 Mhand Hifi and Mustapha Michrafy. Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem. *Computers & operations research*, 34(9):2657–2673, 2007. doi:10.1016/J.COR.2005.10.004.

20 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. doi:10.1016/J.JCSS.2007.06.019.

21 Thiago Alcântara Luiz, Haroldo Gambini Santos, and Eduardo Uchoa. Cover by disjoint cliques cuts for the knapsack problem with conflicting items. *Oper. Res. Lett.*, 49(6):844–850, 2021. doi:10.1016/j.orl.2021.10.001.

22 Carlo Mannino, Gianpaolo Oriolo, Federico Ricci-Tersenghi, and L. Sunil Chandran. The stable set problem and the thinness of a graph. *Oper. Res. Lett.*, 35(1):1–9, 2007. doi:10.1016/J.ORL.2006.01.009.

23 Ulrich Pferschy and Joachim Schauer. The knapsack problem with conflict graphs. *J. Graph Algorithms Appl.*, 13(2):233–249, 2009. doi:10.7155/jgaa.00186.

24 Ulrich Pferschy and Joachim Schauer. Approximation of knapsack problems with conflict and forcing graphs. *J. Comb. Optim.*, 33(4):1300–1323, 2017. doi:10.1007/s10878-016-0035-7.

25 Vijay V Vazirani. *Approximation algorithms*, volume 1. Springer, 2001.

26 Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

27 David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

28 Takeo Yamada, Seija Kataoka, and Kohtaro Watanabe. Heuristic and exact algorithms for the disjunctively constrained knapsack problem. *Information Processing Society of Japan Journal*, 43(9), 2002.