Goebbels, Steffen; Gurski, Frank; Komander, Dominique

The knapsack problem with special neighbor constraints

Mathematical Methods of Operations Research

Provided in Cooperation with:
Springer Nature

Suggested Citation: Goebbels, Steffen; Gurski, Frank; Komander, Dominique (2021) : The knapsack problem with special neighbor constraints, Mathematical Methods of Operations Research, ISSN 1432-5217, Springer, Berlin, Heidelberg, Vol. 95, Iss. 1, pp. 1-34, https://doi.org/10.1007/s00186-021-00767-5

Mathematical Methods of Operations Research (2022) 95:1–34
https://doi.org/10.1007/s00186-021-00767-5

ORIGINAL ARTICLE

The knapsack problem with special neighbor constraints

Steffen Goebbels[1] · **Frank Gurski**[2] · **Dominique Komander**[2]

Received: 19 June 2021 / Revised: 22 September 2021 / Accepted: 25 November 2021 /
Published online: 28 December 2021

Abstract

The knapsack problem is one of the simplest and most fundamental NP-hard problems in combinatorial optimization. We consider two knapsack problems which contain additional constraints in the form of directed graphs whose vertex set corresponds to the item set. In the one-neighbor knapsack problem, an item can be chosen only if at least one of its neighbors is chosen. In the all-neighbors knapsack problem, an item can be chosen only if all its neighbors are chosen. For both problems, we consider uniform and general profits and weights. We prove upper bounds for the time complexity of these problems when restricting the graph constraints to special sets of digraphs. We discuss directed co-graphs, minimal series-parallel digraphs, and directed trees.

Keywords Knapsack problem · Neighbor constraints · Directed co-graphs · Minimal series-parallel digraphs · Directed trees

Mathematics Subject Classification 05C85 · 90C39 · 05C69

A short version of this paper will appear in the proceedings of the International Conference on Operations Research (OR Goebbels et al. 2021).

✉ Frank Gurski
frank.gurski@hhu.de

Steffen Goebbels
steffen.goebbels@hsnr.de

Dominique Komander
dominique.komander@hhu.de

[1] Faculty of Electrical Engineering and Computer Science, iPattern Institute, Niederrhein University of Applied Sciences, 47805 Krefeld, Germany

[2] Institute of Computer Science, Algorithmics for Hard Problems Group, University of Düsseldorf, 40225 Düsseldorf, Germany

1 Introduction

In recent years, the interest in knapsack problems related to graphs has grown strongly. In addition to an input to a knapsack problem, these problems contain additional constraints in the form of graphs whose vertex set corresponds to the item set of the knapsack problem. These include, for example, the knapsack problem with conflict or forcing graphs, the subset sum problem with digraph restrictions, the partially ordered knapsack problem. In Borradaile et al. (2012), applications for constrained knapsack problems in scheduling, tool management, investment strategies, database storage, and network formation are mentioned.

Following Borradaile et al. (2011, 2012), we analyze the knapsack problem with additional constraints based on a digraph.

We use the notations of Bang-Jensen and Gutin (2009) for graphs and digraphs.[1] A *graph* is a pair $G = (A, E)$, where A is a finite set of *vertices* and $E \subseteq \{\{u, v\} \mid u, v \in A,\ u \neq v\}$ is a finite set of *edges*. For a vertex $v \in A$, the set $N_G(v) = \{u \in A \mid \{v, u\} \in E\}$ is the *set of all neighbors* or the *neighborhood* of v in G.

A *directed graph* or *digraph* is a pair $G = (A, E)$, where A is a finite set of *vertices* and $E \subseteq \{(u, v) \mid u, v \in A,\ u \neq v\}$ is a finite set of ordered pairs of distinct vertices called *arcs* or *directed edges*. For a vertex $v \in A$, the sets $N_G^+(v) = \{u \in A \mid (v, u) \in E\}$ and $N_G^-(v) = \{u \in A \mid (u, v) \in E\}$ are called the *set of all successors* and the *set of all predecessors* of v in G. Their union $N_G(v) = N_G^-(v) \cup N_G^+(v)$ is the *set of all neighbors* or the *neighborhood* of v in G. A vertex v is called a *sink* iff $N_G^+(v) = \emptyset$, v is called a *source* iff $N_G^-(v) = \emptyset$. For an undirected graph $G = (A, E)$, replacing every edge $\{u, v\}$ of G by exactly one of the arcs (u, v) and (v, u) leads to an *orientation* of G. Every digraph that can be obtained by an orientation of an undirected graph G is called an *oriented graph*, i.e., an oriented graph is a digraph without loops or opposite arcs.

A digraph $G' = (A', E')$ is a *subdigraph* of digraph $G = (A, E)$ if $A' \subseteq A$ and $E' \subseteq E \cap (A' \times A')$. If every arc of E with start- and end-vertex in A' is in E', we say that G' is an *induced subdigraph* of digraph G.

For a digraph $G = (A, E)$ its *underlying undirected graph* is defined by disregarding the directions of the arcs, i.e. $\mathrm{un}(G) = (A, \{\{u, v\} \mid (u, v) \in E, u, v \in A\})$.

We analyze knapsack problems with digraph constraints for special classes of digraphs: directed trees, directed co-graphs, and minimal series parallel digraphs (msp-digraphs).

A *directed tree* is a digraph for which the underlying undirected graph is a tree. On directed trees, bidirectional edges are allowed.

An *out-rooted tree* (*in-rooted tree*) is a directed tree with a distinguished root but without bidirectional edges such that all vertices can be reached from the root (the root can be reached from all vertices), i.e., all arcs point away from (towards) the root.

A *binary tree* is a rooted tree in which every vertex has at most two children. A *directed binary tree* is a rooted directed tree such that the rooted underlying undirected tree is a binary tree.

[1] Please note that we use A for the vertex set of graphs and digraphs, since it corresponds to the item set of a knapsack instance.

Directed co-graphs and *msp-digraphs* are recursively defined digraphs, see Sects. 3.1 and 4.1. Starting with single vertex digraphs, larger digraphs are constructed by some operations that compute the union of the sets of vertices and the sets of edges of the given digraphs, respectively, and add certain additional edges. The recursive structure of these digraphs can be represented by a tree that allows for dynamic programming.

Within the *knapsack problem (KP)* there is given a set $A = \{a_1, \ldots, a_n\}$ of $n \geq 1$ items. Every item a_j has a size s_j and a profit p_j. Further, there is a capacity c. All values are assumed to be non-negative integers and $s_j \leq c$ for every $j \in \{1, \ldots, n\}$. The task is to choose a subset A' of A, such that $p(A') := \sum_{a_j \in A'} p_j$ is maximized and the *capacity constraint* holds, i.e.

$$s\left(A'\right) := \sum_{a_j \in A'} s_j \leq c. \tag{1}$$

The *subset sum problem (SSP)* is the special case of the knapsack problem for which $p_j = s_j$.

We consider the knapsack problem with additional constraints in the form of (di)graphs $G = (A, E)$ from Borradaile et al. (2011, 2012).

The *one-neighbor constraint* prescribes that an item $v \in A$ can be chosen into $A' \subseteq A$ only if at least one of its neighbors in $N_G(v)$ is chosen, i.e.,

$$\left(v \in A' \wedge N_G(v) \neq \emptyset\right) \Rightarrow N_G(v) \cap A' \neq \emptyset. \tag{2}$$

The *all-neighbors constraint* prescribes that an item $v \in A$ can be chosen into $A' \subseteq A$ only if all its neighbors in $N_G(v)$ are chosen, i.e.,

$$v \in A' \Rightarrow N_G(v) \subseteq A'. \tag{3}$$

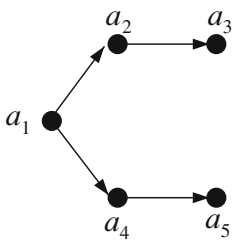
These definitions imply that vertices without neighbors can always be chosen. When considering digraphs, the constraints only apply to the out-neighbors $N_G^+(v)$ of a vertex v. We state the following optimization problems given in Borradaile et al. (2012).

Name Knapsack with one-neighbor constraint (KP1N)
Instance A set $A = \{a_1, \ldots, a_n\}$ of $n \geq 1$ items and a digraph $G = (A, E)$. Every item a_j has a size s_j and a profit p_j. Further, there is a capacity c.
Task Find a subset A' of A that maximizes $p(A')$ subject to (1) and (2).

Name Knapsack with all-neighbors constraint (KPaN)
Instance A set $A = \{a_1, \ldots, a_n\}$ of $n \geq 1$ items and a digraph $G = (A, E)$. Every item a_j has a size s_j and a profit p_j. Further, there is a capacity c.
Task Find a subset A' of A that maximizes $p(A')$ subject to (1) and (3).

As said before, the parameters s_j, p_j and c are assumed to be non-negative integers, i.e. from the set $\mathbb{N}_0 := \{0, 1, 2, 3, \ldots\}$. Especially zero profits p_j need special treatment in some of the presented algorithms.

Fig. 1 Digraph in Example 1



For both problems, a subset A' of A is called *feasible*, if it satisfies the prescribed constraints of the problem. Within some algorithms we need to omit capacity restriction (1) but then we speak nonetheless of feasible solutions. We denote the value of an optimal solution on input I by $\mathrm{OPT}(I)$.

The restriction of KP1N and KPaN to uniform sizes and profits ($s_j = p_j = 1$ for $j = 1, \ldots, n$) is denoted by *uniform* KP1N and *uniform* KPaN, respectively. When general sizes and profits $s_j, p_j \in \mathbb{N}_0$ are allowed as in the definitions of KP1N and KPaN, we also speak of *general* KP1N and *general* KPaN.

Following (Borradaile et al. 2012) we consider the following eight problems

$$\{\text{one-neighbor, all-neighbors}\} \times \{\text{general, uniform}\} \times \{\text{undirected, directed}\}.$$

Example 1 We consider the digraph G in Fig. 1. For uniform KPaN with $c = 3$ we have $\{a_3\}$, $\{a_5\}$, $\{a_3, a_5\}$, $\{a_2, a_3\}$, $\{a_4, a_5\}$, $\{a_2, a_3, a_5\}$, and $\{a_3, a_4, a_5\}$ as feasible solutions. For uniform KP1N with $c = 3$, we additionally find the feasible solutions $\{a_1, a_2, a_3\}$ and $\{a_1, a_4, a_5\}$.

Every feasible solution for knapsack with all-neighbors constraint (KPaN) is also a feasible solution for knapsack with one-neighbor constraint (KP1N), but not vice versa. Both $A' = \emptyset$ and $A' = A$ for $s(A) \leq c$ are feasible solutions for every instance of knapsack with all-neighbors constraint (KPaN) and for every instance of knapsack with one-neighbor constraint (KP1N).

In Tables 1, 2, 3 and 4 we summarize the results of the given paper as well as the known time complexities of knapsack problems with neighbor constraints. Most presented algorithms for general profits and weights are pseudo-polynomial. Their runtimes depend on the capacity bound c or the profit sum

$$P = \sum_{j=1}^{n} p_j \leq n \cdot \max_{1 \leq j \leq n} p_j. \tag{4}$$

Value P is an upper bound for the profit of an optimal solution for an instance of the knapsack problem. There are two similar approaches to solve knapsack with dynamic programming: One can either partition target capacities and maximize profits or one can partition profits and minimize capacities. The first approach leads to pseudo-polynomial runtime bounds in c whereas the second approach results in bounds that

involve P (cf. Kellerer et al. 2010, Chapter 2). Within this paper, we use the second approach.

The general, directed, all-neighbors knapsack problem is closely related to the partially ordered knapsack problem (POK) (Johnson and Niemi 1983; Kolliopoulos and Steiner 2007), which is defined as follows. Given is an instance for the knapsack problem and an acyclic digraph $G = (A, E)$ whose vertices correspond to the items of the knapsack instance (cf. Sect. 4). A subset $A' \subseteq A$ is *closed under predecessor*, if $v \in A'$ and $(u, v) \in E$ implies that $u \in A'$. The partially ordered knapsack problem is to find a subset $A' \subseteq A$ which is closed under predecessor, such that A' maximizes the profit and fulfills the capacity constraint (1). In contrast to this, solutions of the all-neighbors knapsack problem are closed under successor. By considering the reverse graph, one constraint on an acyclic digraph becomes the other. Since we discuss the all-neighbors knapsack problem even on cyclic digraphs, this is a generalization of partially ordered knapsack.

The subset-union knapsack problem (SUKP) is a special case of KPaN, see (Borradaile et al. 2012). KP1N can be applied to solve the *knapsack problem with forcing graph* (KFG): In the KFG problem, for all edges of an undirected forcing graph (A, E), at least one of its two vertices has to be chosen as an item into a feasible KP solution, see (Pferschy and Schauer 2017). Let P be the sum of all profits. To model the KFG constraint with a one-neighbor constraint in a graph (A', E'), let the set of vertices be $A' := A \cup E$ such that edges of the forcing graph become vertices/items with size 0 and profit $P + 1$, and vertices/items of A keep their size and profit in the KP1N problem that also shares the capacity bound with the KFG problem. Each vertex in $v \in E \subseteq A'$ is connected with two directed edges (pointing) to the two adjacent vertices of the edge v in the forcing graph. If the profit of an optimal KP1N solution is less than $|E| \cdot (P + 1)$, then no feasible solution of KFG exists. Otherwise, the optimal profit of KFG equals the optimal profit of KP1N minus $|E| \cdot (P + 1)$.

Gourvès et al. introduced a related approach, namely the subset sum problem with digraph constraint (SSG) and subset sum problem with weak digraph constraint (SSGW) (Gourvès et al. 2018). SSG is a general KPaN problem on a digraph for which item sizes and profits are equal. Then capacity bound c also becomes a bound for the profit. Originally, the digraph-constraint in SSG is formulated somewhat different: A feasible solution A' of SSG has to fulfill a predecessor condition: If a vertex v of the digraph has at least one predecessor in A', then v must also be in A'. But this constraint equals the all-neighbors constraint from the perspective of the predecessors. We therefore can verify known runtime bounds (Gurski et al. 2020a, b) for SSG with the bounds that we derive for general KPaN on directed co-graphs and msp-digraphs. A dynamic program to solve SSG for oriented trees is presented in Gourvès et al. (2018). This can be also done with the dynamic program for the KPaN problem in Sect. 5.1. SSGW, however, is different to the problems discussed in this paper. The digraph constraint here is that if all predecessors of a vertex v are in a feasible solution, then v must also be in this solution. Bounds for this problem on directed co-graphs and msp-digraphs are proved in Gurski et al. (2020a, b).

APX-hard problems do not allow a PTAS (and thus no FPTAS), if P $\neq$ NP. For subset selection problems, the existence of a pseudo-polynomial algorithm leads to a FPTAS. Thus, for APX-hard subset selection problems there is no pseudo-polynomial

Table 1 Time complexity of knapsack problems with neighbor constraints

	Graph	One-neighbor		All-neighbors	
Uniform	Undirected	Linear	Borradaile et al. (2012)	$\mathcal{O}(n \cdot c) \subseteq \mathcal{O}(n^2)$	Borradaile et al. (2012)
	Directed	Strongly NP-hard	Borradaile et al. (2012)	Strongly NP-hard	Borradaile et al. (2012)
General	Undirected	APX-hard	Borradaile et al. (2012)	PFTAS	Borradaile et al. (2012)
				$\mathcal{O}(n \cdot P + n^2)$	Conclusions
	Directed	Strongly NP-hard	Borradaile et al. (2012)	Strongly NP-hard	Borradaile et al. (2012)

Table 2 Time complexity of knapsack problems with neighbor constraints given by trees

	Graph	One-neighbor		All-neighbors	
Uniform	Undirected	Linear	Borradaile et al. (2012)	$\mathcal{O}(1)$	One component
	Directed	$\mathcal{O}(n^3)$	Theorem 6	$\mathcal{O}(\mathrm{n}^3)$	Theorem 5
General	Undirected	NP-hard	Proposition 2	$\mathcal{O}(n)$	Proposition 1
	Directed	NP-hard	Proposition 2	NP-hard	Proposition 2
		$\mathcal{O}(n \cdot P^2 + n)$	Theorem 6	$\mathcal{O}(n \cdot (P+1) \cdot (P+n))$	Theorem 5

Table 3 Time complexity of knapsack problems with neighbor constraints given by co-graphs

	Graph	One-neighbor		All-neighbors	
Uniform	Undirected	Linear	Borradaile et al. (2012)	$\mathcal{O}(n \cdot c) \subseteq \mathcal{O}(n^2)$	Borradaile et al. (2012)
	directed	$\mathcal{O}(n^3)$	Corollary 3	$\mathcal{O}(n^3)$	Corollary 2
General	Undirected	$\mathcal{O}(n \cdot P^2+n^2)$	Conclusions	$\mathcal{O}(n \cdot P+n^2)$	Conclusions
	Directed	$\mathcal{O}(n \cdot P^2+n^2)$	Theorem 2	$\mathcal{O}(n \cdot (P+1) \cdot \max\{n, P+1\})$	Theorem 1

Table 4 Time complexity of knapsack problems with neighbor constraints given by msp-digraphs

	One-neighbor		All-neighbors	
Uniform	$\mathcal{O}(n^3)$	Corollary 5	$\mathcal{O}(n^3)$	Corollary 4
General	$\mathcal{O}(n \cdot P^2 + n^2)$	Theorem 4	$\mathcal{O}(n \cdot (P+1) \cdot \max\{n, P+1\})$	Theorem 3

algorithm, i.e. they are strongly NP-hard. In the Conclusions section we show that knapsack with one-neighbor constraint and knapsack with all-neighbors constraint are pseudo-polynomial subset selection problems if zero profits are excluded. Thus, these problems are not APX-hard.

In the following sections we discuss solutions for the one-neighbor and all-neighbors knapsack problems for which the input graph is restricted to directed co-graphs, msp-digraphs, and directed trees. We consider general profits and weights. Then, results for uniform profits and weights immediately follow.

2 Elementary results

In order to show the correctness of our runtime bounds for KP1N and KPaN when restricting the graph constraints to special sets of digraphs, we will use the following results.

Lemma 1 *Let $G = (V_G, E_G)$ be a digraph, and let $H = (V_H, E_H)$ be an induced subdigraph of G. If A' is a feasible solution for KPaN on G, then $A' \cap V_H$ is a feasible solution for KPaN on H.*

Proof Let $v \in A' \cap V_H$. We have to show that $N_H^+(v) \subseteq A' \cap V_H$. Since A' is a feasible solution for KPaN on G, we know that $N_G^+(v) \subseteq A'$. Thus, $N_H^+(v) \subseteq N_G^+(v) \subseteq A'$, and $N_H^+(v) = N_H^+(v) \cap V_H \subseteq N_G^+(v) \subseteq A' \cap V_H$. □

The reverse direction of Lemma 1 does not hold, since vertices with successors in $A' \cap (V_G \setminus V_H)$ are not considered by the feasible solutions for KPaN on H. By considering the induced subdigraph $H = (\{a_1, a_4, a_5\}, \{(a_1, a_4), (a_4, a_5)\})$ of digraph G and $A' = \{a_1, a_2, a_3\}$ in Example 1, we observe that Lemma 1 does not hold for KP1N.

But we can show a weaker form of Lemma 1 for KP1N:

Lemma 2 *Let $G = (V_G, E_G)$ be a digraph and let $H = (V_H, E_H)$ be an induced subdigraph of G, such that no non-sink of H has a successor in $V_G \setminus V_H$. If A' is a feasible solution for KP1N on G, then $A' \cap V_H$ is a feasible solution for KP1N on H.*

Proof Let $v \in A' \cap V_H$ with $N_H^+(v) \neq \emptyset$, i.e., v is not a sink in H. We have to show $N_H^+(v) \cap (A' \cap V_H) \neq \emptyset$.

Due to the preliminaries, non-sink v does not have a successor in $V_G \setminus V_H$, i.e., $N_G^+(v) \subseteq V_H$. Thus, $\emptyset \neq N_H^+(v) = N_G^+ \cap V_H = N_G^+(v)$. Since $v \in A'$, and A' is a feasible solution for KP1N on G, and $N_G^+(v) \neq \emptyset$, there exists $u \in A' \cap N_G^+(v) = A' \cap N_H^+(v) = A' \cap N_H^+(v) \cap V_H$. But this gives $N_H^+(v) \cap (A' \cap V_H) \neq \emptyset$. □

Lemma 3 *Let $G = (V_G, E_G)$ be a digraph and let $H = (V_H, E_H)$ be an induced subdigraph of G. If A' is a feasible solution for KP1N on G and $A' \subseteq V_H$, then A' is a feasible solution for KP1N on H.*

Proof Let $v \in A'$ with $N_H^+(v) \neq \emptyset$. We have to show $N_H^+(v) \cap A' \neq \emptyset$.

Since A' is a feasible solution on G and $\emptyset \neq N_H^+(v) \subseteq N_G^+(v)$, we know that $N_G^+(v) \cap A' \neq \emptyset$. Because of $A' \subseteq V_H$, we get $A' = A' \cap V_H$, such that $\emptyset \neq N_G^+(v) \cap A' = N_G^+(v) \cap V_H \cap A' = N_H^+(v) \cap A'$. □

The connected components can be used to solve the general, undirected all-neighbors knapsack problem.

Proposition 1 *General knapsack with all-neighbors constraint (KPaN) is solvable in $\mathcal{O}(t \cdot P + n + m)$ time on graphs with n vertices, m edges, and $t \leq n$ connected components with the capacity bound c. If a graph has only one component, e.g., a tree, then the problem is solvable in $\mathcal{O}(n)$ time.*

Proof The given problem can be interpreted as a knapsack problem in which the items are $t \leq n$ connected components of the graph, cf. (Borradaile et al. 2012). Computing the connected components can be done in $\mathcal{O}(n + m)$ time. Size and profit of each component is the sum of sizes and profits of its vertices. These data also can be computed in $\mathcal{O}(n + m)$ time. With t items, the knapsack problem can be solved in $\mathcal{O}(t \cdot c)$ and in $\mathcal{O}(t \cdot P)$ time with standard dynamic programming, cf. (Kellerer et al. 2010, Chapter 2). Thus, the general all-neighbors problem on (undirected) graphs is solvable in $\mathcal{O}(t \cdot P + n + m)$ time. If the graph consists of only one single component, then it is sufficient to sum up profits and sizes, the runtime is in $\mathcal{O}(n)$. □

When restricting the graph constraints to edgeless (di)graphs, both general knapsack with one-neighbor constraint and with all-neighbors constraint become the NP-hard knapsack problem such that they are also NP-hard.

Co-graphs, directed co-graphs (see Definition 1) as well as msp-digraphs (see Definition 2) are allowed to be edgeless (di)graphs. Thus, we get:

Corollary 1 *General knapsack with one-neighbor constraint (KP1N) and general knapsack with all-neighbors constraint (KPaN) are NP-hard on co-graphs, directed co-graphs as well as on msp-digraphs.*

Proposition 2 *General knapsack with one-neighbor constraint (KP1N) and general knapsack with all-neighbors constraint (KPaN) are NP-hard for directed in-rooted trees and out-rooted trees. General knapsack with one-neighbor constraint (KP1N) is NP-hard for (undirected) trees.*

Proof We use a reduction from KP. For some given KP instance I on n items $B = \{b_1, \ldots, b_n\}$ with a capacity bound d, where item b_i has size t_i and profit q_i, $1 \leq i \leq n$, we define instances I' as follows. The item set is $A = \{a_i \mid b_i \in B\} \cup \{a_{n+1}\}$ where item a_i has size $s_i := t_i$ and profit $p_i := q_i$, $1 \leq i \leq n$, and a_{n+1} has size $s_{n+1} = 0$ and profit $p_{n+1} = 0$. The capacity is $c = d$, and the digraph is $G = (V, E)$ with $V = A$. For in-rooted trees, we choose $E = \{(a_i, a_{n+1}) \mid 1 \leq i \leq n\}$, for

out-rooted trees let $E = \{(a_{n+1}, a_i) \mid 1 \leq i \leq n\}$, and for undirected trees let $E = \{\{a_i, a_{n+1}\} \mid 1 \leq i \leq n\}$.

Obviously, I has a KP solution $B' \subseteq B$ if and only if I' has a solution $\{a_i \mid b_i \in B'\} \cup \{a_{n+1}\}$ or $\{a_i \mid b_i \in B'\}$. Note that a_{n+1} is not allowed to be in non-trivial solutions of I' for out-rooted trees and KPaN. It is also not included for KP1N on (undirected) trees or out-rooted trees iff $B' = \emptyset$. □

3 Knapsack problems on directed co-graphs

3.1 Directed co-graphs

Directed co-graphs are interesting from an algorithmic point of view since several hard digraph problems can be solved in polynomial time by dynamic programming along the tree structure of the input digraph, see (Bang-Jensen and Maddaloni 2014; Gurski 2017; Gurski et al. 2019a, b, 2020, 2021; Gurski and Rehs 2018; Retoré 1998). Moreover, directed co-graphs are very useful for the reconstruction of the evolutionary history of genes or species using genomic sequence data (Hellmuth et al. 2017; Nojgaard et al. 2018).

Definition 1 (*Directed co-graphs*, Crespelle and Paul 2006) The class of *directed co-graphs* is recursively defined as follows.

(i) Every digraph $(\{v\}, \emptyset)$ on a single vertex, denoted by v, is a *directed co-graph*.
(ii) If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are vertex-disjoint directed co-graphs, then

(a) the *disjoint union* $G_1 \oplus G_2$, which is defined as the digraph with vertex set $V_1 \cup V_2$ and edge set $E_1 \cup E_2$,
(b) the *order composition* $G_1 \oslash G_2$, defined by their disjoint union plus all possible edges only directed from V_1 to V_2, and
(c) the *series composition* $G_1 \otimes G_2$, defined by their disjoint union plus all possible edges between V_1 and V_2 in both directions,

are *directed co-graphs*.

Undirected *co-graphs* are defined accordingly with the operations "disjoint union" and "series composition" $G_1 \otimes G_2$ that adds all edges between V_1 and V_2, see (Corneil et al. 1981).

A directed co-graph can be represented by an expression X that connects vertices using the three operations of Definition 1, see Example 2. The expression X is called a *di-co-expression*, digraph(X) is the defined graph, and we denote by $|X|$ the number of vertices in digraph(X).

Obviously, we can define a tree structure for every directed co-graph, denoted as *di-co-tree*. The leaves of the di-co-tree represent the vertices of the digraph and the inner nodes of the di-co-tree correspond to the operations applied on the sub-expressions defined by the subtrees. For some vertex u of di-co-tree T we denote by $T(u)$ the subtree rooted at u and by $X(u)$ the di-co-expression (*sub-di-co-expression*, *sub-expression*) defined by $T(u)$.

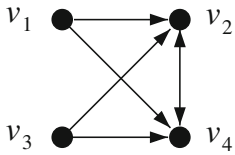


Fig. 2 Digraph in Example 2

Given some directed co-graph, one can construct a di-co-tree in linear time, see (Crespelle and Paul 2006). It has been shown in Crespelle and Paul (2006) that directed co-graphs can be characterized by eight forbidden induced subdigraphs.

Example 2 The di-co-expression

$$X = ((v_1 \oplus v_3) \oslash (v_2 \otimes v_4)) \tag{5}$$

defines digraph(X) shown in Fig. 2.

Several classes of digraphs are included in the set of all directed co-graphs. For example, every transitive tournament is a directed co-graph, and every oriented bipartite digraph is a directed co-graph.

3.2 All-neighbors problems on directed co-graphs

We consider an instance I of the general knapsack with all-neighbors constraint problem such that $G = (V, E)$ is a directed co-graph defined by some binary di-co-expression X.

For modeling a specific restriction, the series composition is not required in all-neighbors problems. A directed co-graph can be replaced by a simplified directed co-graph for which the di-co-expression does not contain a series composition. As none or all vertices have to be chosen in a feasible solution, each subexpression $L \otimes R$ can be replaced by a single vertex for which the size is the sum of all sizes and the profit is the sum of all profits in $L \otimes R$.

For some sub-expression $X(u)$ of X let $F(X(u), p)$ be the minimum size of a solution with profit exactly p in digraph$(X(u))$. While the problem requires the size to be at most c, values of function F are allowed to exceed c. But if profit p cannot be realized under digraph restrictions, we set $F(X(u), p) := \infty$.

Theorem 1 *General KPaN can be solved in directed co-graphs with n vertices in $\mathcal{O}(n(P+1)\max\{n, P+1\})$ time where $P := \sum_{j=1}^{n} p_j$ is the sum of all profits of the items (vertices) of a given problem, see Eq. (4).*

Proof In order to solve the general KPaN problem for an instance I on directed co-graph G, we traverse di-co-tree T in a bottom-up order, i.e. from the leaves to the root r. For every vertex u of T and integer $0 \le p \le P$ we use Algorithm 1 to compute a non-negative integer value $F(X(u), p)$ as described before.

If a zero profit $p = 0$ has to be realized then $F(X(u), p) = 0$ due to the feasible empty solution. Non-empty zero profit solutions cannot have smaller sizes.

If $|X(u)| = 1$ and $p > 0$ then $F(X(u), p) < \infty$ iff the profit p_i of the only vertex a_i equals p. Then $F(X(u), p) = s_i$, otherwise $F(X(u), p) = \infty$.

To prove the correctness for $|X(u)| > 1$ and $p > 0$, we look at the three possible operations:

- Let $X(u) = L \oplus R$. Since there are no edges between digraph(L) and digraph(R), each feasible solution (a solution that fulfills the all-neighbors condition without capacity constraint) is the union of a feasible solution with vertices in digraph(L) and a feasible solution with vertices in digraph(R), see Lemma 1, and vice versa since neighborhoods do not change by applying the union:

$$F(L \oplus R, p) = \min\{F(L, p') + F(R, p - p') : 0 \leq p' \leq p\}.$$

- Let $X(u) = L \oslash R$. If a feasible solution has at least one vertex in digraph(L) then it includes all vertices of digraph(R) due to the $\oslash$-operation and the all-neighbors condition. Thus, a solution equals a solution for digraph(R) without vertices in digraph(L), i.e, $F(X(u), p) = F(R, p)$, or it includes at least one vertex of digraph(L). In the latter case, the all-neighbor condition implies, that the restriction of the solution to digraph(L) is also a feasible solution (Lemma 1). Vice versa, by adding all vertices of digraph(R), each feasible solution in digraph(L) becomes a solution fulfilling the all-neighbors condition in digraph($L \oslash R$). Let S_R be the sum of all sizes for vertices in digraph(R), and let P_R be the corresponding sum of all profits of digraph(R). As mentioned above, solutions that include at least one vertex of digraph(L) can only exist if $P_R \leq p$. Then, for solutions including a vertex of L, it holds

$$F(X(u), p) = F(L, p - P_R) + S_R.$$

- Let $X(u) = L \otimes R$. Since $p > 0$, we only have to deal with non-empty solutions. The only possible feasible solution contains all vertices. If, for example, digraph(L) $\neq \emptyset$ then due to the $\otimes$-operation and the all-neighbors condition, all vertices of non-empty digraph(R) also belong to the solution. For one of these vertices, "$\otimes$" in connection with the all-neighbors condition also requires all vertices of digraph(L) to be part of the solution. Thus, the solution includes all vertices. Vice versa, let S_X be the sum of all sizes for vertices in digraph($X(u)$), and let P_X be the corresponding sum of all profits of digraph($X(u)$). If $p = P_X$ then $F(X(u), p) = S_X$, otherwise $F(X(u), p) = \infty$.

After performing Algorithm 1 on every sub-expression $X(u)$ for vertices u of T, we can solve our problem by considering values for root r:

$$\mathrm{OPT}(I) = \max\{p \in \{0, \ldots, P\} \mid F(X(r), p) \leq c\}.$$

A di-co-tree T with n vertices and m edges can be computed in $\mathcal{O}(n + m) \subseteq \mathcal{O}(n^2)$ time, see (Crespelle and Paul 2006), and P is computed in $\mathcal{O}(n)$ time. We have n leaves and $n-1$ inner vertices in di-co-tree T for which values of F for $p \in \{0, \ldots, P\}$ have to be computed with at most $P+1$ iterations or by adding n summands. Thus, the runtime is in $\mathcal{O}(n(P + 1)\max\{n, (P + 1)\} + n + m) \subseteq \mathcal{O}(n(P + 1)\max\{n, P + 1\})$. □

For uniform problems, P equals the number of items n. This is the reason why uniform problems can be solved in polynomial time instead of the pseudo-polynomial bounds which hold for the general problems. Theorem 1 directly implies:

Corollary 2 *Uniform KPaN can be solved on directed co-graphs with n vertices in $\mathcal{O}(n^3)$ time.*

Algorithm 1 General all-neighbors problems on directed co-graphs: Compute minimal solution size $F(X(u), p)$ for a given profit p based on previously computed values for sub-expressions of $X(u)$.

▷ Start Block "Common"

$F(X(u), p) := \infty$
$(V, E) :=$ digraph$(X(u))$
if $p = 0$ **then** $F(X(u), p) := 0$
else if $|X(u)| = 1$ **then** let $a_i \in V$ be the only vertex.
  if $p = p_i$ **then** $F(X(u), p) := s_i$
else if $X(u) = L \oplus R$ **then**
  for $p' := 0; p' \leq p; p' := p' + 1$ **do**
    $S := F(L, p') + F(R, p - p')$
    if $F(X(u), p) > S$ **then** $F(X(u), p) := S$

▷ End Block "Common"

if $p \neq 0 \wedge (X(u) = L \oslash R \vee X(u) = L \otimes R)$ **then**
  $(V_L, E_L) :=$ digraph(L), $(V_R, E_R) :=$ digraph(R)
  $P_L := \sum_{a_i \in V_L} p_i$, $P_R := \sum_{a_i \in V_R} p_i$
  $S_L := \sum_{a_i \in V_L} s_i$, $S_R := \sum_{a_i \in V_R} s_i$
  if $X(u) = L \oslash R$ **then**
    if $p \leq P_R$ **then** $F(X(u), p) := F(R, p)$
    else $F(X(u), p) := S_R + F(L, p - P_R)$
  else ▷ (case $X(u) = L \otimes R$)
    if $p = P_L + P_R$ **then** $F(X(u), p) = S_L + S_R$

We can apply the result to the subset sum problem with digraph constraint (SSG) on directed co-graphs. If we introduce an upper capacity bound $c = P' \leq P$ for profits such that instead of finding a subset A' of A that maximizes $p(A')$ the task is to find a subset A' of A that maximizes $\{p' : p' = p(A') \wedge p' \leq P'\}$, then the runtime bound of KPaN becomes $\mathcal{O}(n(P' + 1) \max\{n, P' + 1\})$. If we also exclude zero profits, then the sums giving P_L, P_R, S_L, and S_R in Algorithm 1 can be limited to the first $P' + 1$ summands. Then the runtime bound becomes

$$\mathcal{O}\left(n\left(P' + 1\right)^2 + n^2\right) \subseteq \mathcal{O}\left(n\left(P'\right)^2 + n^2\right).$$

Taking into account that the number m of arcs is bounded by n^2, this is the same estimation as in Gurski et al. (2020a). In fact, the result in Gurski et al. (2020a) is proved following the recursive definition of directed co-graphs very similarly to the structure of the dynamic program presented in this section.

3.3 One-neighbor problems on directed co-graphs

We consider an KP1N instance I of the general problem such that G is a directed co-graph defined by some binary di-co-expression X.

The directed co-graph G can possess cycles and does not necessarily contain sinks or sources. Therefore, a feasible, non-empty KP1N solution does not necessarily contain sinks of G. Such a feasible KP1N solution without sinks with respect to digraph(L) is also a feasible solution with respect to digraph($L \oslash R$) since every vertex within the solution already has a successor in digraph(L) that belongs to the solution. But this is not true for feasible KP1N solutions with sinks of digraph(L).

In order to get useful informations about the sinks within a solution, we use an extended data structure for dynamic programming. For some sub-expression $X(u)$ of X let $F(X(u), p, k)$ be the minimum size of a feasible solution fulfilling the one-neighbor condition with profit exactly p in digraph($X(u)$) that contains a sink if $k = 1$ and does not possess any sinks if $k = 0$. The minimum size is allowed to exceed c, i.e., as before, feasible solutions must fulfill the one-neighbor constraint but do not have to fulfill the capacity constraint. We set $F(X(u), p, k)$ to ∞, whenever there is no such a solution.

Profits (and sizes) of vertices a_i are allowed to be zero. Thus, a zero profit can be realized with either an empty solution or with non-empty solutions that only contain vertices a_i with zero profit $p_i = 0$. In contrast to empty solutions, vertices with zero profit can be used to fulfill neighbor constraints (for example, see Algorithm 3, case $X = L \oslash R$ for $p'' = 0$, and case $X = L \otimes R$ for $p' = 0$ or $p' = p$; also see Algorithm 5 for $p' = p$). To differentiate between empty and non-empty zero profit solutions, we introduce function $\tilde{F}(X(u), p, k)$ that is defined as $F(X(u), p, k)$ with the difference that now instead of feasible solutions only non-empty feasible solutions are considered. If there are only empty feasible solutions then $\tilde{F}(X(u), p, k) = \infty$. Thus,

$$F(X(u), p, k) = \begin{cases} \tilde{F}(X(u), p, k), & \text{if } p > 0 \vee k = 1, \\ 0, & \text{else.} \end{cases}$$

Theorem 2 *General KP1N can be solved in directed co-graphs with n vertices in $\mathcal{O}(P^2 n + n^2)$ time.*

Proof While traversing di-co-tree T with root r of a directed co-graph G in a bottom-up order, we compute $\tilde{F}(X(u), p, k)$ for every vertex u of T and integers $0 \leq p \leq P$, $k \in \{0, 1\}$ with Algorithm 3.

If there exists only one vertex a_i, i.e., $|X(u)| = 1$, then this vertex is a sink: if $p = p_i$ then $\tilde{F}(X(u), p, 1) = s_i$. In all other cases $\tilde{F}(X(u), p, k) = \infty$.

To see the correctness for $|X(u)| > 1$ and $0 \leq p \leq P$, we look at the three possible operations:

– Let $X(u) = L \oplus R$. A feasible solution in digraph($L \oplus R$) restricted to digraph(L) or digraph(R) is still feasible with respect to the subdigraphs due to Lemma 2. Feasible solutions of the two subdigraphs are also feasible solutions in digraph($L \oplus$

Algorithm 2 Given a di-co-expression $X(u)$, the dynamic knapsack program computes the minimal size $\hat{F}(X(u), p)$ of a non-empty set of vertices (with sizes and profits) from the vertices of digraph(X) such that their profits exactly sum up to $p \in \{0, \ldots, P\}$.

if $|X(u)| = 1$ **then** let $a_i \in V$ be the only vertex.
  if $p = p_i$ **then** $\hat{F}(X(u), p) := s_i$
  else $\hat{F}(X(u), p) := \infty$
else
  X has one of the representations $X = L \oplus R$, $X = L \oslash R$ or $X = L \otimes R$
  $\hat{F}(X(u), p) := \min\{\hat{F}(L, p), \hat{F}(R, p)\}$ ▷ empty solutions for $p' = 0$ in R or L, respectively
  for $p' = 1$; $p' < p$; $p' := p' + 1$ **do**
    $S := \hat{F}(L, p') + \hat{F}(R, p - p')$
    if $S < \hat{F}(X(u), p)$ **then** $\hat{F}(X(u), p) := S$

R) as the neighbors do not change. But then, the one-neighbor condition is fulfilled for their union as well.

For $k = 0$, a feasible solution must not have a sink. This is true iff both feasible sub-solutions with vertices in digraph(L) and digraph(R), respectively, do not have sinks. For $k = 1$, at least one sub-solution must contain a sink.

Function $\tilde{F}$ deals with non-empty solutions. A feasible solution in digraph$(L \oplus R)$ is non-empty iff its restriction to digraph(L) or its restriction to digraph(R) is non-empty. This leads to equations

$$\begin{aligned}
&\tilde{F}(X(u), p, 0) = \min\{\tilde{F}(L, p, 0), \tilde{F}(R, p, 0),\\
&\quad \min\{\tilde{F}(L, p', 0) + \tilde{F}(R, p - p', 0) \mid p' \in \{1 \ldots, p - 1\}\}\},\\
&\tilde{F}(X(u), p, 1) = \min\{\tilde{F}(L, p, 1), \tilde{F}(R, p, 1),\\
&\quad \min\{\tilde{F}(L, p', k') + \tilde{F}(R, p - p', k'') \mid p' \in \{1, \ldots, p - 1\}, k', k'' \in \{0, 1\}, k' + k'' \geq 1\}\}.
\end{aligned}$$

Note that the separate listing of $\tilde{F}(L, p, 0)$, $\tilde{F}(R, p, 0)$ in the first equation and of $\tilde{F}(L, p, 1)$, $\tilde{F}(R, p, 1)$ in the second equation is necessary to compose non-empty sub-solutions with empty sub-solutions to get non-empty solutions.

– Let $X(u) = L \oslash R$.

Because of this case, the sink parameter k was introduced. If a non-empty solution is completely contained in digraph(L) then it cannot contain a sink since the $\oslash$-operation and the neighbor condition imply that the sink must include a neighbor in digraph(R) which is in the solution. On the other side, a sink-free feasible solution with respect to digraph(L) is also a feasible solution for digraph$(L \oslash R)$.

It remains to discuss the case of feasible solutions with at least one vertex in digraph(R). Since there are no edges from digraph(R) to digraph(L), the restriction of the solution to digraph(R) still fulfills the neighbor condition (Lemma 2). Vice versa, if a non-empty feasible solution with respect to digraph(R) is given, then due to the $\oslash$-operation, each vertex of digraph(L) has a successor in this solution and can therefore be added without violating the one-neighbor condition.

To sum up, each set S of vertices from digraph$(L \oslash R)$ satisfies the one-neighbor condition if and only if the following conditions hold: The restriction of S to digraph(R) fulfills the one neighbor condition. If this restriction is empty then

the restriction of S to digraph(L) fulfills the one-neighbor condition and does not possess a sink. Otherwise, if the restriction of S to digraph(R) is non-empty, S can contain arbitrary vertices from digraph(L).
With $\hat{F}(L, p)$ we compute the smallest size of a non-empty knapsack solution without any digraph restrictions and without capacity bound (i.e., a subset sum solution) that obtains exactly profit p, see Algorithm 2. Let V_L be the set of vertices of digraph(L). If $V_L = \{a_1, \ldots, a_n\}$ and a_j has size s_j and profit p_j, then

$$\hat{F}(L, p) := \min \left\{ \sum_{a_j \in V'} s_j \mid \emptyset \neq V' \subseteq V_L,\ p = \sum_{a_j \in V'} p_j \right\}$$

if the set is non-empty, otherwise $\hat{F}(L, p) := \infty$. Then, with

$$\gamma(L, R, p, k) := \min \left\{ \tilde{F}(R, p, k),\ \min \left\{ \tilde{F}(R, p'', k) + \hat{F}(L, p - p'') \mid 0 \leq p'' < p \right\} \right\}, \tag{6}$$

it holds for $\tilde{F}(L \oslash R, p, k)$:

$$\tilde{F}(L \oslash R, p, 0) = \min \left\{ \tilde{F}(L, p, 0), \gamma(L, R, p, 0) \right\},\ \tilde{F}(L \oslash R, p, 1) = \gamma(L, R, p, 1). \tag{7}$$

Note that in (6) value p'' is allowed to be zero. Thus, non-empty solutions with zero profit are used to fulfill the one-neighbor condition. The expression $\tilde{F}(L, p, 0)$ in (7) is necessary to cover the case of a non-empty solution without vertices in digraph(R).

– Let $X(u) = L \otimes R$. Since the $\otimes$-operation generates a digraph without sinks, feasible solutions can only exist for $k = 0$, i.e., $\tilde{F}(L \otimes R, p, 1) = \infty$. For $k = 0$, we seek a solution with minimal size and have to consider different profits.
With

$$\rho(L, R, p) := \min \left\{ \hat{F}(L, p') + \hat{F}(R, p - p') \mid 0 \leq p' \leq p \right\},$$

we get

$$\tilde{F}(L \otimes R, p, 0) = \min \left\{ \tilde{F}(L, p, 0), \tilde{F}(R, p, 0), \rho(L, R, p) \right\}.$$

All non-empty solutions with vertices both in digraph(L) and digraph(R) are covered with the calculation of $\rho(L, R, p)$. In this situation, the one-neighbor condition is fulfilled due to the "$\otimes$" operator. Non-empty solutions without vertices in digraph(L) or digraph(R) are considered via $\tilde{F}(R, p, 0)$ or $\tilde{F}(L, p, 0)$, respectively.

We can solve our problem by considering values at root r of T:

$$\mathrm{OPT}(I) = \max\{p \in \{0, \ldots, P\} \mid p = 0 \vee \exists_{k \in \{0,1\}} \tilde{F}(X(r), p, k) \leq c\}.$$

Before we perform Algorithm 3 on every sub-expression $X(u)$ for vertices u of T, we compute values of $\hat{F}$ with Algorithm 2 on every sub-expression $X(u)$ for $\mathcal{O}(n)$ vertices and every $p \in \{0, \ldots, P\}$ by executing a loop with $\mathcal{O}(P)$ iterations. Thus, all values of $\hat{F}$ are computed in $\mathcal{O}(nP^2 + n)$ time.

Similar to Algorithms 2, 3 is executed for $0 \leq p \leq P$ and each of the $\mathcal{O}(n)$ nodes of the di-co-tree. The loops within the algorithm have at most $p+1$ iterations with $p \leq P$. As before, the di-co-tree of a directed co-graph can be computed in $\mathcal{O}(n^2)$ time, and P is computed in $\mathcal{O}(n)$ time. This gives an overall runtime of $\mathcal{O}((P+1)^2 n + n^2) \subseteq \mathcal{O}(P^2 n + n^2)$. This bound also covers the runtime of Algorithm 2. □

With $P = n$, the result for uniform KP1N follows directly:

Corollary 3 *Uniform KP1N can be solved for directed co-graphs on n vertices in $\mathcal{O}(n^3)$ time.*

4 Knapsack problems on minimal series-parallel digraphs

4.1 Minimal series-parallel digraphs

Minimal Series-parallel digraphs are interesting from an algorithmic point of view since several hard graph problems can be solved in polynomial time by dynamic programming along the tree structure of the input graph, see (Gurski et al. 2020, 2021; Valdes et al. 1982).

For $n \geq 2$, the digraph

$$\overrightarrow{P_n} = (\{v_1, \ldots, v_n\}, \{(v_i, v_{i+1}) \mid 1 \leq i \leq n-1\})$$

is called a *directed path* on n (distinct) vertices. Vertex v_1 is the *start vertex* and v_n is the *end vertex* of $\overrightarrow{P_n}$. For $n \geq 2$, the digraph

$$\overrightarrow{C_n} = (\{v_1, \ldots, v_n\}, \{(v_i, v_{i+1}) \mid 1 \leq i \leq n-1\} \cup \{(v_n, v_1)\})$$

is a *directed cycle* on n vertices. A *directed acyclic graph* is a digraph without any directed cycle as subdigraph. A vertex v is *reachable* from vertex u in G, if there is a directed path in G as a subdigraph with start vertex u and end vertex v.

We recall the definitions from (Bang-Jensen and Gutin, 2018) which are based on Valdes et al. (1982).

Definition 2 (*Minimal series-parallel digraphs*) The class of *minimal series-parallel digraphs*, *msp-digraphs* for short, is recursively defined as follows.

(i) Every digraph $(\{v\}, \emptyset)$ on a single vertex, denoted by v, is a *minimal series-parallel digraph*.

(ii) If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are vertex-disjoint minimal series-parallel digraphs, then the parallel composition $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$ is a *minimal series-parallel digraph*.

Algorithm 3 General KP1N on directed co-graphs: Compute minimal solution size $\tilde{F}(X(u), p, k)$ for a given profit p based on previously computed values for sub-expressions of $X(u)$.

▷ Start block "Common"

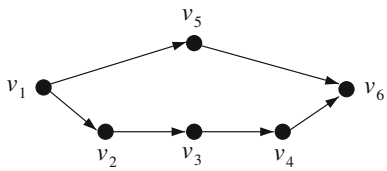
$\tilde{F}(X(u), p, k) := \infty$
if $|X(u)| = 1$ **then** let $a_i \in V$ be the only vertex.
  if $k = 1 \wedge p = p_i$ **then** $\tilde{F}(X(u), p, k) := s_i$
else if $X(u) = L \oplus R$ **then**
  ▷ Consider empty and non-empty solutions with zero profit in R or L
  $\tilde{F}(X(u), p, k) := \tilde{F}(L, p, k)$
  if $\tilde{F}(X(u), p, k) > \tilde{F}(R, p, k)$ **then** $\tilde{F}(X(u), p, k) := \tilde{F}(R, p, k)$
  ▷ Consider solutions with positive profit in L and R
  for $p' = 1; p' < p; p' := p' + 1$ **do**
    if $k = 0$ **then**
      $S := \tilde{F}(L, p', 0) + \tilde{F}(R, p - p', 0)$
      if $\tilde{F}(X(u), p, k) > S$ **then** $\tilde{F}(X(u), p, k) := S$
    else
      $S_1 := \tilde{F}(L, p', 1) + \tilde{F}(R, p - p', 0)$, $S_2 := \tilde{F}(L, p', 0) + \tilde{F}(R, p - p', 1)$
      $S_3 := \tilde{F}(L, p', 1) + \tilde{F}(R, p - p', 1)$
      for $i = 1; i \leq 3; i := i + 1$ **do**
        if $\tilde{F}(X(u), p, k) > S_i$ **then** $\tilde{F}(X(u), p, k) := S_i$
▷ End block "Common"
if $X(u) = L \oslash R$ **then**
  if $k = 0$ **then** $\tilde{F}(X(u), p, k) := \tilde{F}(L, p, k)$
  if $\tilde{F}(X(u), p, k) > \tilde{F}(R, p, k)$ **then** $\tilde{F}(X(u), p, k) := \tilde{F}(R, p, k)$
  for $p'' = 0; p'' < p; p'' := p'' + 1$ **do**
    $S_R := \tilde{F}(R, p'', k)$
    if $S_R < \infty$ **then**
      $S_L := \hat{F}(L, p - p'')$, see Algorithm 2
      if $\tilde{F}(X(u), p, k) > S_L + S_R$ **then** $\tilde{F}(X(u), p, k) := S_L + S_R$
else if $X(u) = L \otimes R$ **then** ▷ digraph$(X(u))$ has no sinks
  if $k = 0$ **then**
    $\tilde{F}(X(u), p, k) := \tilde{F}(L, p, 0)$, $S_R := \tilde{F}(R, p, 0)$
    if $S_R < \tilde{F}(X(u), p, k)$ **then** $\tilde{F}(X(u), p, k) := S_R$
    ▷ Consider solutions with at least one vertex from both digraphs without restrictions
    for $p' = 0; p' \leq p; p' := p' + 1$ **do**
      $S_L := \hat{F}(L, p')$, see Algorithm 2
      $S_R := \hat{F}(R, p - p')$, see Algorithm 2
      if $S_L + S_R < \tilde{F}(X(u), p, k)$ **then** $\tilde{F}(X(u), p, k) := S_L + S_R$

(iii) If G_1 and G_2 are vertex-disjoint minimal series-parallel digraphs and O_1 is the set of sinks in G_1 and I_2 is the set of sources in G_2, then series composition $G_1 \times G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup (O_1 \times I_2))$ is a *minimal series-parallel digraph*.

Regarding these construction rules, new edges only result from rule (iii). Thus, an msp-digraph is a directed, acyclic graph that has at least one source and one sink.

Similar to a directed co-graph, an msp-digraph can be represented by an *msp-expression* X that connects vertices using the operations of Definition 2, see Example 3. The digraph defined by X is denoted with digraph(X), and $|X|$ is the number of vertices in digraph(X). For every msp-digraph we can define a tree structure, denoted as *msp-tree*. The leaves of the msp-tree represent the vertices of the digraph and the inner

Fig. 3 Digraph in Example 3



nodes of the msp-tree correspond to the operations. For some vertex u of msp-tree T we denote by $T(u)$ the subtree rooted at u and $X(u)$ the msp-expression (*sub-msp-expression*, *sub-expression*) defined by $T(u)$. For every msp-digraph one can construct an msp-tree in linear time (Valdes et al. 1982).

Minimal series-parallel digraphs are the line digraphs of edge series-parallel digraphs (Valdes et al. 1982), which are an oriented version of the well known class of series-parallel graphs. In order to motivate the notation of *minimal* series-parallel digraphs, we refer to the super class of series-parallel digraphs which are are exactly the digraphs whose transitive closure equals the transitive closure of an msp-digraph (Valdes et al. 1982).

Example 3 The msp-expression

$$L = (v_1 \times ((v_2 \times (v_3 \times v_4)) \cup v_5)) \times v_6$$

defines the msp-digraph shown in Fig. 3.

Several classes of digraphs are included in the set of all msp-digraphs. For example, every in- and out-rooted tree is an msp-digraph, and every oriented bipartite graph is an msp-digraph.

The following lemma can be immediately obtained via induction on the recursive definition of minimal series-parallel digraphs.

Lemma 4 *Let $G = (V, E)$ be an msp-digraph. Then, for every vertex $v \in V$ there is a sink v_s of G, such that there is a directed path from v to v_s in G, and there is a source v_o of G, such that there is a path from v_o to v in G.*

For assembling feasible solutions from sub-expressions, we use the following lemma.

Lemma 5 *Let G be an msp-digraph. Then every non-empty feasible solution of KP1N and every non-empty feasible solution of KPaN contains a sink of G.*

Proof Let $G = (V, E)$ be an msp-digraph and A' be a feasible KP1N (KPaN) solution which contains some $v \in V$. There exits a longest path in the directed acyclic graph induced by A' containing v. Obviously, this path ends in a sink of the induced digraph. But due to the neighbor conditions, this sink has to be a sink of G, too. □

4.2 All-neighbors problems on minimal series-parallel digraphs

We consider an instance I of the general knapsack problem with all-neighbors constraint problem such that G is an msp-digraph defined by some binary msp-expression

X. Since msp-digraphs are acyclic, the problem becomes a partially ordered knapsack problem by considering the reverse digraph.

Theorem 3 *General KPaN can be solved in msp-digraphs with n vertices in $\mathcal{O}(n(P+1)\max\{n,(P+1)\})$ time.*

Proof The arguments are similar to the proof of Theorem 1 when we consider msp-expressions instead of di-co-expressions.

Let T be an msp-tree for msp-digraph G with root r that we traverse from the leaves to the root. For every vertex u of T and profit $0 \leq p \leq P$ we compute the minimal size $F(X(u), p)$ of feasible solutions (that fulfill the all-neighbor condition but might exceed the capacity bound c) with respect to digraph$(X(u))$ for profit exactly p with Algorithm 4.

For cases $p = 0$, $|X(u)| = 1$, and $X(u) = L \cup R$ we refer to the proof of Theorem 1 since "$\oplus$" and "$\cup$" are the same operations. Thus, we only have to deal with $X(u) = L \times R$ for $0 < p \leq P$.

If a solution with respect to digraph$(L \times R)$ contains a vertex of digraph(L) then, due to the all-neighbors condition and since digraph(L) is acyclic, it also contains a sink of digraph(L), see Lemma 5. Therefore, all sources in digraph(R) belong to the solution. Since digraph(R) is an acyclic digraph, all its vertices are reachable from sources (see Lemma 4). Because of the all-neighbors condition, all $|R|$ vertices of digraph(R) belong to the solution as well. The remaining vertices in digraph(L) fulfill the all-neighbors condition in digraph$(L \times R)$ and therefore also fulfill the all-neighbors condition in digraph(L) (Lemma 1). Vice versa, a feasible solution for digraph(R) is also a feasible solution for digraph$(L \times R)$ since the vertices in the solution do not have any neighbors (successors) in digraph(L). Each feasible solution for digraph(L) can be extended to a feasible solution for digraph$(L \times R)$ by adding all vertices of digraph(R) such that the all-neighbor condition holds (especially for sinks with respect to digraph(L) that get successors belonging to the solution).

Thus, let S_R be the sum of all sizes and P_R be the sum of all profits corresponding to vertices of digraph(R). If $p \leq P_R$, then there exists a smallest solution iff a smallest solution is contained in digraph(R) and $F(L \times R, p) = F(R, p)$. Otherwise, if $p > P_R$, all vertices of digraph(R) belong to every solution:

$$F(L \times R, p) = \min\{F(L, p - P_R) + S_R\}.$$

After performing Algorithm 4 on every sub-expression $X(u)$ of X the problem is solved via

$$\mathrm{OPT}(I) = \max\{p \in \{0, \ldots, P\} \mid F(X(r), p) \leq c\}.$$

with the same arguments as in the proof of Theorem 1, the runtime is in $\mathcal{O}(n(P+1)\max\{n, P+1\})$. □

We can apply the result to the subset sum problem with digraph constraint (SSG) for msp-digraphs and capacity bound $c = P' \leq P$ in the same way as for directed co-graphs. The runtime bound of KPaN becomes $\mathcal{O}(n(P'+1)\max\{n, P'+1\})$. If

one also excludes zero profits, then the sums giving S_L and P_R in Algorithm 4 can be limited to the first $P' + 1$ summands, and the runtime is in $\mathcal{O}(n(P')^2 + n^2)$. This confirms the result in Gurski et al. (2020a) that was obtained with a similar dynamic program.

For $P = n$, Theorem 3 directly implies:

Corollary 4 *Uniform KPaN can be solved on msp-digraphs with n vertices in* $\mathcal{O}(n^3)$ *time.*

Algorithm 4 General all-neighbors problems on msp-digraphs: Compute minimal solution size $F(X(u), p)$ based on previously computed values for sub-expressions of $X(u)$.

Execute Block "Common" in Algorithm 1 with "$\oplus$" replaced by "$\cup$"
if $X(u) = L \times R$ **then**
  $(V_R, E_R) := \text{digraph}(R)$
  $S_R := \sum_{a_i \in V_R} s_i$, $P_R := \sum_{a_i \in V_R} p_i$
  if $p \leq P_R$ **then** $F(X(u), p) := F(R, p)$
  else
    $S_L := F(L, p - P_R)$
    if $S_L + S_R < F(X(u), p)$ **then** $F(X(u), p) := S_L + S_R$

4.3 One-neighbor problems on minimal series-parallel digraphs

The induced digraph of every non-empty KPaN or KP1N solution on an msp-digraph G is an acyclic digraph. Therefore, it contains a sink. Due to the neighbor constraint, all its sinks are also sinks of G (see Lemma 5). Obviously, it also contains at least one source, but the sources of the induced digraph might not be sources of G. Since the "$\times$" operation connects sinks with sources of msp-digraphs, we have to distinguish between solutions with and without sources of the underlying graph.

In order to get useful information about the sources within a solution, we use extended data structures similar to the discussion of the general KP1N problem on directed co-graphs. We consider an instance of KP1N such that $G = (A, E)$ is an msp-digraph which is given by an msp-expression X. For some sub-expression $X(u)$ of X let $F(X(u), p, k)$ be the minimum size of a feasible solution fulfilling the one-neighbor condition with profit exactly p in digraph$(X(u))$ that contains a source if $k = 1$ and that does not possess a source if $k = 0$. The minimum size is allowed to exceed c, i.e., as before, feasible solutions must fulfill the one-neighbor constraint but do not necessarily fulfill a capacity constraint. We set $F(X(u), p, k)$ to ∞, whenever there is no such a solution.

As for KP1N on directed co-graphs, we also need to distinguish between empty and non-empty solutions. For this purpose, we use the same function $\tilde{F}(X, p, k)$ which only considers non-empty feasible solutions such that $\tilde{F}(X, p, k) = \infty$ if only the empty solution exists for $p = 0$.

Theorem 4 *General KP1N can be solved on msp-digraphs with n vertices in* $\mathcal{O}(P^2n + n^2)$ *time.*

Proof The arguments are similar to the proof of Theorem 2 when we consider msp-expressions instead of di-co-expressions and indicate sources instead of sinks.

Let T be an msp-tree for msp-digraph G with root r that we traverse from the leaves to the root. For every vertex u of T and profit $0 \leq p \leq P$ we compute $\tilde{F}(X(u), p, k)$ to decide if a non-empty solution for profit p exists that has to contain sources in digraph$(X(u))$ for $k = 1$ and must not contain sources in digraph$(X(u))$ for $k = 0$. For this, we use Algorithm 5 and discuss its correctness.

For cases $|X(u)| = 1$ and $X(u) = L \cup R$ see proof of Theorem 2 as "$\oplus$" and "$\cup$" are the same operations, and there is no difference between indication of sinks or sources. Thus, we only have to deal with $X(u) = L \times R$ for $0 \leq p \leq P$. In the following, we show that each set of vertices satisfying the one-neighbor condition fulfills one of the following conditions, and vice versa.

1. It has no source in digraph$(L \times R)$ and its vertices all belong to digraph(R) and it is a feasible solution in digraph(R).
2. It is composed from vertices of both digraph(L) and digraph(R) and the restrictions to digraph(L) and digraph(R) fulfill the one-neighbor condition in digraph(L) and digraph(R), respectively. The solution contains a vertex that is a source of digraph(R). It contains a source of digraph$(L \times R)$ iff it contains a source of digraph(L).

We prove that feasible solutions fulfill one of the conditions. Let a solution with $(k = 1)$ or without $(k = 0)$ sources with respect to digraph$(L \times R)$ be given. Note that due to the "$\times$" operation there are no sources in digraph(R) with respect to digraph$(L \times R)$ since at least one existing sink of acyclic digraph(L) is connected with all sources of digraph(R).

Thus, only for the case $k = 0$ it is possible that all vertices of the solution belong to digraph(R). This describes the first condition. In this case, the neighbor condition is already fulfilled in digraph(R) (Lemma 3).

If not all vertices of the solution belong to digraph(R), at least one vertex of the solution is from digraph(L). Then the non-empty restriction of the solution to digraph(L) satisfies the one-neighbor condition with respect to digraph(L) as all vertices with successors in digraph(R) become sinks (Lemma 2). The restriction of the solution to digraph(R) also fulfills the one-neighbor condition with Lemma 2 since vertices in digraph(R) do not have successors in digraph(L). There has to be at least one solution vertex in digraph(R) that is a source in digraph(R) since otherwise existing vertices of the solution that are sinks with respect to digraph(L) (see Lemma 5) do not fulfill the one-neighbor condition in digraph$(L \times R)$. This is condition 2 in which sources can only be in digraph(L).

Now we prove that each set of vertices satisfying the two conditions is a feasible solution in digraph$(L \times R)$.

If a feasible solution exists with respect to digraph(R) (condition 1), then it is also a feasible solution with respect to digraph$(L \times R)$ (without sources) since the neighbors of the vertices of the solution do not change. Non-empty feasible solutions with respect

to digraph(L) and with respect to digraph(R) (condition 2) can be combined to a feasible solution for digraph($L \times R$) since the condition 2 prescribes that the solution with respect to digraph(R) contains a source in digraph(R). The source is necessary to fulfill the one-neighbor condition for the existing sinks of the solution in digraph(L), see Lemma 5.

The two conditions lead to following equations. With

$$\gamma(L, R, p, k) := \min\{\tilde{F}(L, p', k) + \tilde{F}(R, p - p', 1) \mid 0 \leq p' \leq p\}, \tag{8}$$

it holds

$$\tilde{F}(L \times R, p, 0) = \min\{\tilde{F}(R, p, 0), \tilde{F}(R, p, 1), \gamma(L, R, p, 0)\},$$
$$\tilde{F}(L \times R, p, 1) = \gamma(L, R, p, 1).$$

Note that the expressions $\tilde{F}(R, p, 0)$, $\tilde{F}(R, p, 1)$ are required to cover source-free solutions without vertices in digraph(L). The value $p' = 0$ in (8) has to be considered in the case $k = 1$ to include solutions with a source that do not have non-zero profit vertices in L. A source must be contained in L.

After performing Algorithm 5 on every sub-expression $X(u)$ of X we can solve our problem:

$$\mathrm{OPT}(I) = \max\{p \in \{0, \ldots, P\} \mid p = 0 \vee \exists_{k \in \{0,1\}} \tilde{F}(X, p, k) \leq c\}.$$

with the same arguments as in the proof of Theorem 2, the runtime is in $\mathcal{O}(P^2 n + n^2)$.□

With $P = n$ we immediately get:

Corollary 5 *Uniform KP1N can be solved on msp-digraphs with n vertices in $\mathcal{O}(n^3)$ time.*

Algorithm 5 General one-neighbor problems on msp-digraphs: Compute minimal solution size $\tilde{F}(X(u), p, k)$ based on previously computed values for sub-expressions of $X(u)$. A solution without source in digraph($X(u)$) is required for $k = 0$, a solution with source is investigated for $k = 1$. As in Algorithm 3, function $\tilde{F}(X(u), p, k)$ only computes minimal sizes of non-empty solutions. If the only solution for $p = 0$ is empty then $\tilde{F}(X(u), p, k) := \infty$.

Execute Block "Common" in Algorithm 3 with "$\oplus$" replaced by "$\cup$"
if $X(u) = L \times R$ **then**
  if $k = 0$ **then**
    $\tilde{F}(X(u), p, k) := \min\{\tilde{F}(R, p, 0), \tilde{F}(R, p, 1)\}$
  for $p' = 0; p' \leq p; p' := p' + 1$ **do**
    $S_L := \tilde{F}(L, p', k), S_R := \tilde{F}(R, p - p', 1)$
    if $S_L + S_R < \tilde{F}(X, p, k)$ **then** $\tilde{F}(X(u), p, k) := S_L + S_R$

5 Knapsack problems on directed trees

In this section, we discuss digraphs for which the underlying undirected graph is a tree, i.e., directed trees. Examples are in-and out-rooted trees that are msp-digraphs but generally not directed co-graphs. Thus, we have already shown upper bounds for in-and out-rooted trees, see Table 4. Now we are interested in the general case in which edges can be directed arbitrarily. This includes the case of *oriented trees*, i.e., directed trees that are oriented digraphs, but also covers trees with bidirectional edges. General KPaN and KP1N on directed trees are NP-hard, see Proposition 2.

The presented bounds in Table 2 are given for algorithms that traverse an underlying undirected tree from the leaves to the root. They compute several sizes for each subtree based on the values obtained for previously handled subtrees.

5.1 All-neighbors problems on directed trees

When dealing with the all-neighbors condition, vertices that are connected with bidirectional edges must be included in a solution together or not at all. Thus, before solving all-neighbors problems, each bidirectional edge can be removed by replacing their vertices with a single new one. For a directed tree with $\mathcal{O}(n)$ vertices, this can be done in $\mathcal{O}(n)$ time by traversing the underlying undirected tree. Profits and sizes of the new vertices being created are the sum of the profits and sizes of merged vertices, respectively. The resulting directed tree also has $\mathcal{O}(n)$ vertices.

Theorem 5 *General KPaN can be solved on directed trees with n vertices in $\mathcal{O}(n(P+1)(P+n))$ time, uniform KPaN can be solved on directed trees in $\mathcal{O}(n^3)$ time.*

For msp-digraphs and thus for in- and out-rooted trees we have an $\mathcal{O}(n^3)$ runtime bound for uniform KPaN. This fits with the given bound for more general oriented and even directed trees that are not necessarily msp-digraphs (as e.g. the induced subdigraph that results from removing v_3 in Fig. 3).

Proof Let $T = (V, E)$ be a directed tree where we choose an arbitrary vertex $r \in V$ as root. Due to the previous remark, we can assume without loss of generality that there are no bidirectional edges in T. For $u \in V$ we denote by $T(u)$ the subtree of T rooted at vertex u. We now distinguish between out-going and in-coming edges of u. Let $S_+(u) = \{o_1, \ldots, o_{\delta^+(u)}\}$ be the set of neighbor vertices in $T(u)$ that can be reached from u with an out-going edge, and let $S_-(u) = \{i_1, \ldots, i_{\delta^-(u)}\}$ be the set of neighbor vertices in $T(u)$ from which the root u can be reached with an in-coming edge. These pairwise disjoint sets can be computed in $\mathcal{O}(n)$ time from T. The order of the two sets is denoted by $\delta(u) = \delta^+(u) + \delta^-(u)$ and corresponds to the vertex degree of u in the underlying undirected tree $\text{un}(T)$ minus 1, since we omit the predecessor of u in T.

With $F(T(u), p, k)$, we compute the minimum size of a solution on subtree $T(u)$ subject to the all-neighbors constraint with profit exactly p that includes the root u for $k = 1$ and does not include the root if $k = 0$. The values are allowed to exceed c. If no solution exists, $F(T(u), p, k) := \infty$.

To compute $F(T(u), p, k)$ based on the (previously computed) values of $F(T(v), p, 0)$ and $F(T(v), p, 1)$ for $v \in S_+(u) \cup S_-(u)$, we need to represent $q = p$ or $q = p - p_u$ (if $p \geq p_u$), where p_u is the profit of u, by sums of $\delta(u)$ numbers $(q_1, q_2, \ldots, q_{\delta(u)})$, $q_j \in \{0, \ldots, q\}$. If $\delta(u) > 0$, there exist

$$\binom{(q+1) + (\delta(u) - 1) - 1}{\delta(u) - 1} \leq (q+n)^{\delta(u)-1}$$

different sequences of such numbers that sum up to q. Let $K(q)$ be the set of these sequences, and $K(q) := \emptyset$ if $\delta(u) = 0$.

Lemma 1 allows to assemble all-neighbors solutions from such solutions on subtrees with dynamic programming. We begin with solutions in $T(u)$ that do not include the root. Due to the all-neighbors condition, such solutions are not allowed to contain vertices that are connected with the root by in-coming edges. If $\delta(u) = 0$, i.e., u is a leaf of the underlying undirected tree $\mathrm{un}(T)$, then the minimum size of all these solutions is $F(T(u), p, 0) = 0$ if $p = 0$, and $F(T(u), p, 0) = \infty$ if $p \neq 0$. For $\delta(u) > 0$ we get

$$F(T(u), p, 0) = \min \left\{ \sum_{o_j \in S_+(u)} \min\{F(T(o_j), q_j, 0), F(T(o_j), q_j, 1)\} + \sum_{i_j \in S_-(u)} F(T(i_j), q_{j+\delta^+(u)}, 0) \mid (q_1, q_2, \ldots, q_{\delta(u)}) \in K(p) \right\}.$$

Let s_u be the size of u. If $p < p_u$ then $F(T(u), p, 1) = \infty$. Otherwise, all solutions including the root u contain vertices that can be directly reached from the root with an out-going edge. If $\delta(u) = 0$ then $F(T(u), p, 1) = s_u$ for $p = p_u$ and $F(T(u), p, 1) = \infty$ otherwise. If $\delta(u) > 0$,

$$F(T(u), p, 1) = s_u + \min \left\{ \sum_{o_j \in S_+(u)} F(T(o_j), q_j, 1) + \sum_{i_j \in S_-(u)} \min\{F(T(i_j), q_{j+\delta^+(u)}, 0), F(T(i_j), q_{j+\delta^+(u)}, 1)\} \mid (q_1, q_2, \ldots, q_{\delta(u)}) \in K(p - p_u) \right\}.$$

For our rooted directed tree we define $\Delta = \max\{\delta(u) \mid u \in V\}$ which corresponds to the maximum vertex degree of $\mathrm{un}(T)$ minus 1. We have to compute values of F for at most $P + 1$ profits and $\mathcal{O}(n)$ vertices $u_1, \ldots, u_n$ with at most Δ summands to find

$$\mathrm{OPT}(I) = \max\{p \in \{0, \ldots, P\} \mid \exists_{k \in \{0,1\}} F(T, p, k) \leq c\}.$$

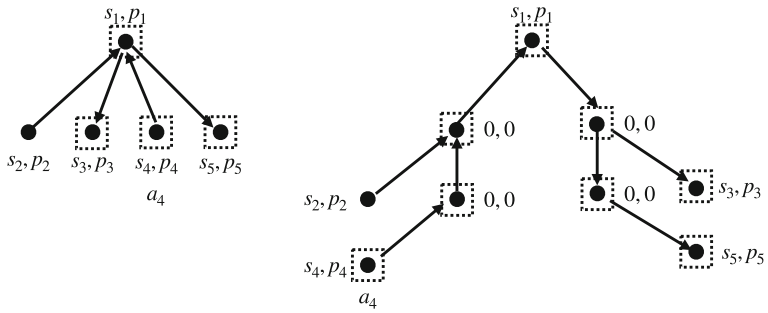


Fig. 4 Transforming an all-neighbors problem on an oriented tree into an all-neighbors problem on a binary tree: Vertex a_4 is chosen into a feasible solution. Then all vertices inserted to obtain the binary tree are also chosen

For the non-trivial case $\Delta > 0$ this leads to a runtime bound

$$\mathcal{O}\left(n + \sum_{p=0}^{P}\sum_{l=1}^{n} \Delta \cdot d(l)\right) \subseteq \mathcal{O}\left(n + (P+1)n\Delta(P+n)^{\Delta-1}\right), \tag{9}$$

where $\mathcal{O}(n)$ is the runtime to compute tree structures and P,

$$d(l) := \begin{cases} (p+n)^{\delta(u_l)-1} & : \delta(u_l) > 1 \\ 1 & : \delta(u_l) \leq 1. \end{cases} \tag{10}$$

To reduce the vertex degree, the general problem allows inserting additional vertices in T with size and profit zero before executing the previously described procedure, see Fig. 4.

For each root u of a subtree with $\delta(u) > 2$, we reduce the number of successors to at most two by inserting separate binary trees for out-going and in-coming edges as shown in the figure. In linear time $\mathcal{O}(n)$, this results in a new directed tree T' with at most $2n$ vertices and $\Delta \leq 2$. Then the all-neighbors constraint ensures that each solution with respect to T becomes a solution with respect to T' containing only possibly some additional new vertices, and vice versa. Thus, we obtain the bound for general KPaN from (9) with $\Delta = 2$.

Instead of solving uniform KPaN, we solve general KPaN on T' with constant weights and sizes for vertices of T and zero weights and size for the new inner vertices of binary trees. Thus, we get the bound of the uniform problem with $P = n$. □

If one does not eliminate all bidirectional edges in a first step, the proof becomes only slightly longer. But then, bidirectional edges can be used to reduce the vertex degree as shown in Fig. 5.

The dynamic program used in Theorem 5 is also applicable to the SSG problem for $c = P'$ on directed trees. By replacing P with P', one obtains the bound $\mathcal{O}(n(P'+1)(P'+n))$. A different dynamic program solving this SSG problem is given in Gourvès et al. (2018).

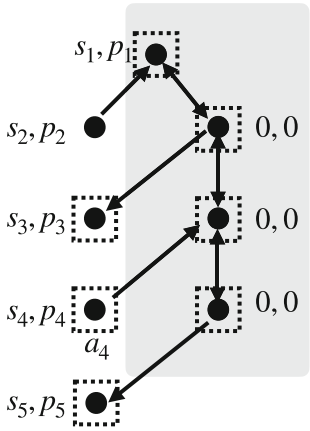


Fig. 5 Transforming an all-neighbors problem on a directed tree from Fig. 4 into an all-neighbors problem on a binary tree by inserting bidirectional edges: Vertex a_4 is chosen into a feasible solution. Then all new vertices inserted to obtain the binary tree are also chosen as one due to the bidirectional edges

5.2 One-neighbor problems on directed trees

In contrast to the all-neighbors problem, a one-neighbor constraint might change by merging vertices that are connected with bidirectional edges.

Theorem 6 *General KP1N can be solved in directed trees T with n vertices in $\mathcal{O}\left(nP^2 + n\right)$ time, uniform KP1N can be solved on directed trees in $\mathcal{O}\left(n^3\right)$ time.*

Proof Let T be a directed tree and as before, let r be an arbitrary chosen root of the underlying undirected tree $\mathrm{un}(T)$. We organize the vertices from root to leaves such that for a vertex u, $S_+(u)$ and $S_-(u)$ are again the vertices which are directly connected from u in the subtree of T rooted at u with an out-going and in-coming edge, respectively. For the one-neighbor problem, we must also consider the disjoint set of neighbor vertices $S_\pm(u) = \{b_1, \ldots, b_{\delta^\pm(u)}\}$ in $T(u)$ that are connected with u by a bidirectional edge. Again, these information can be gathered in $\mathcal{O}(n)$.

We want to solve the problem in a tree with $\Delta \leq 2$ as in the previous proof. However, it is even difficult to realize $\Delta \leq 3$ by switching to an equivalent problem. The one-neighbor condition allows to reduce non-empty sets $S_+(u)$ and $S_-(u)$ to at most one vertex each by inserting directed binary out- or in-rooted trees with root u as before. However, this approach does not work for $S_\pm(u)$. Additionally added bidirectional vertices can be used to fulfill a one-neighbor condition that is not fulfilled for the original tree and bidirectional edges cannot be replaced by out-going and in-coming edges to auxiliary vertices without violating the tree structure. Therefore, we solve the problem on a data structure that avoids this difficulty, see Fig. 6.

The data structure consists of five components (type, L, R, size, profit). Depending on the given type, it represents either an edge $\{u, v\}$ of the underlying undirected tree $\mathrm{un}(T)$, i.e., a one or bidirectional edge of T, or a leaf u of T. A tree with $\Delta \leq 2$ is constructed by recursively inserting this structure into L (left subtree) and R (right subtree). The empty structure () is used to terminate the construction. The right subtrees in R are used to compose a linear list of technical nodes such that structures for all vertices in $S(u) := S_+(u) \cup S_-(u) \cup S_\pm(u)$ can be attached to one individual node

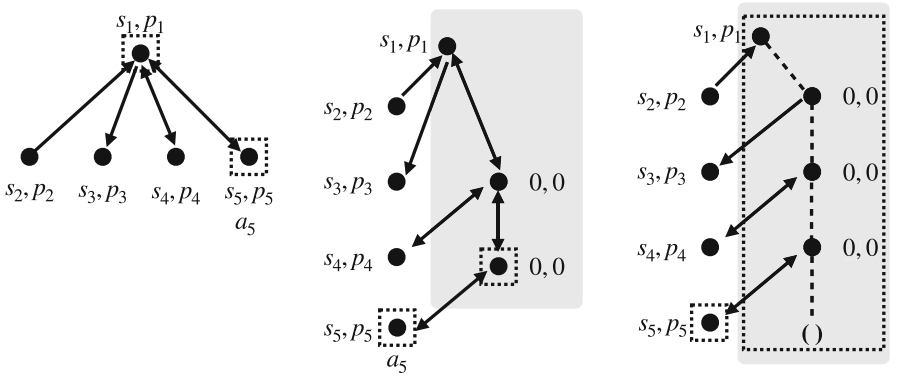


Fig. 6 Transforming a one-neighbor problem on an arbitrary tree into a one-neighbor problem on a tree with vertex degree at most three is difficult due to bidirectional edges. In this example, vertex a_5 is chosen into a feasible solution. Then the construction in the proof of Theorem 5, transferred to bidirectional edges as shown in the middle image, does not work with the one-neighbor constraint. Therefore, we transform into the auxiliary binary tree structure on the right side

with the left subtree field L. The field type indicates the type of object (edge or leaf):

$$\text{type} := \begin{cases} \text{out} & : v \in S_+(u) \\ \text{in} & : v \in S_-(u) \\ \text{inout} & : v \in S_\pm(u) \\ \text{leaf} & : u \text{ is a leaf of } T. \end{cases}$$

Let u be a vertex of the given tree T with size s_u and profit p_u. Then, we generate structure

$$X(u) = (\text{type}, L, R, s_u, p_u)$$

as follows: If $S(u) = \emptyset$ then type := leaf, $L := R := ()$. Otherwise, let $v \in S(u)$, $S'(u) := S(u) \setminus \{v\}$. Field type is set to the type of edge between u and v, $L := X(v)$. If $S'(u) = \emptyset$ then $R := ()$. Otherwise, let $w \in S'(u)$, $S'' := S'(u) \setminus \{w\}$. Then,

$$R := (\text{type of edge between } u \text{ and } w, X(w), R'', 0, 0).$$

To insert the expression for R'' we iterate the procedure with $S'(u)$ replaced by S'', and so on. For example, Fig. 6 shows a tree that is transformed to expression

$$\Big(\text{in}, \Big(\text{leaf}, (), (), s_2, p_2\Big), \Big(\text{out}, \Big(\text{leaf}, (), (), s_3, p_3\Big), \Big(\text{inout}, \Big(\text{leaf}, (), (), s_4, p_4\Big),$$
$$\Big(\text{inout}, (\text{leaf}, (), (), s_5, p_5), (), 0, 0\Big), 0, 0\Big), 0, 0\Big), s_1, p_1\Big).$$

The vertices connected with dashed edges in Fig. 6 represent one vertex of the original tree T. With respect to the one-neighbor condition they have to be interpreted as one.

We derive a structure X from a given tree, but we also need to translate X back into the tree by merging linear lists in fields R (dashed edges) into single vertices that can be incident to multiple edges again. Let $T'(X)$ be the corresponding tree that can be obtained from X. Thus, $T'(X(r))$ is a reconstructed version of tree T, and for each vertex u of T, tree $T'(X(u))$ corresponds to subtree $T(u)$.

We define $F(X, k, p)$ on structure X that computes minimal sizes of corresponding KP1N solutions for profit exactly p without size limit (1) in the tree $T'(X)$ under an additional constraint depending on the value of parameter k:

- $k = 0$ requires that the root of $T'(X)$ is not part of a KP1N solution in $T'(X)$,
- $k = 1$ indicates that the root of $T'(X)$ is in the solution but does not have a neighbor in $T'(X)$ that also belongs to the solution,
- $k = 2$ indicates that the root of $T'(X)$ is in the solution and also has a neighbor in the solution.
- $k = 3$ indicates "semi-solutions" which include the root of $T'(X)$ and for which the one-neighbor condition holds for all vertices except from the root. Such "semi-solutions" might become part of solutions at a later step when the root of $T'(X)$ is connected to a preceding vertex of T. Solutions of cases $k = 1$ and $k = 2$ are included in case $k = 3$.

Given a one-neighbor KP1N solution in T, let X be an expression derived from a subtree of T. Then, the restriction of the solution to $T'(X)$ fulfills in $T'(X)$ one of the cases $k \in \{0, 1, 2, 3\}$. Case $k = 3$ is required since the prerequisites of Lemma 2 might not be fulfilled for the root of $T'(X)$ but for all other vertices. Vice versa, solutions in T can be assembled from sets of vertices that fulfill combinations of cases $k \in \{0, 1, 2, 3\}$. This leads to a dynamic program.

Let $\mathcal{P}_p := \{0, 1, 2, \ldots, p\}$. We start with the case $k = 0$ in which we only discuss solutions that do not include the root of $T'(X)$. Therefore, in order to satisfy the one-neighbor condition in $T'(X)$, the root cannot be used:

$$F((), 0, p) = F((\text{leaf}, (), (), s_u, p_u), 0, p) = \begin{cases} 0 & : p = 0 \\ \infty & : p > 0, \end{cases}$$

$$F((\text{in}, L, R, s_u, p_u), 0, p) = \min\{F(L, k', p') + F(R, 0, p-p') \mid k' \in \{0, 2\}, p' \in \mathcal{P}_p\},$$

$$F((\text{out}, L, R, s_u, p_u), 0, p) = \min\{F(L, k', p') + F(R, 0, p-p') \mid k' \in \{0, 1, 2\}, p' \in \mathcal{P}_p\},$$

$$F((\text{inout}, L, R, s_u, p_u), 0, p) = \min\{F(L, k', p') + F(R, 0, p-p') \mid k' \in \{0, 2\}, p' \in \mathcal{P}_p\}.$$

Let $k \geq 1$, then the root of $T'(X)$ has to be in the solution such that for $p_u > p$ we get $F((\text{type}, L, R, s_u, p_u), k, p) := \infty$. We discuss cases for $p_u \leq p$:

In case $k = 1$, no vertex that is connected from the root by an out-going edge or bidirectional edge is allowed to be in the solution:

$$F((), 1, p) = \begin{cases} 0 & : p = 0 \\ \infty & : p > 0, \end{cases} \quad F((\text{leaf}, (), (), s_u, p_u), 1, p) = \begin{cases} s_u & : p = p_u \\ \infty & : p \neq p_u, \end{cases}$$

$$F((\text{in}, L, R, s_u, p_u), 1, p) = s_u + \min\{F(L, k', p') + F(R, 1, p-p') \mid k' \in \{0, 3\}, p' \in \mathcal{P}_{p-p_u}\},$$

$$F((\text{out}, L, R, s_u, p_u), 1, p) = F((\text{inout}, L, R, s_u, p_u), 1, p) = \infty.$$

With $k' = 3$ in the in-rule, even "semi-solutions" can be connected to the root. Since the root is part of the solution, a possibly violated one-neighbor condition of a "semi-solution" is healed. As $k' = 3$ also covers the cases $k' = 1$ and $k' = 2$, all cases $k' \in \{0, 1, 2, 3\}$ are allowed in $F(L, k', p')$.

If $k = 2$, the root of $T'(X)$ and a vertex that is connected from the root by an out-going edge or bidirectional edge must be in the solution. If this root is represented in X by a linear list (like the dashed list in Fig. 6) then it is sufficient for the root to be connected to one neighbor in T via the list. This leads to following rules:

$$
\begin{aligned}
&F((), 2, p) = F((\text{leaf}, (), (), s_u, p_u), 2, p) = \infty,\\
&F((\text{in}, L, R, s_u, p_u), 2, p) = s_u + \min\{F(L, k', p') + F(R, 2, p - p') \mid k' \in \{0, 3\}, p' \in \mathcal{P}_{p-p_u}\},\\
&F((\text{out}, L, R, s_u, p_u), 2, p) = s_u + \min\{F(L, k', p') + F(R, 3, p - p'),\\
&\qquad\qquad F(L, 0, p') + F(R, 2, p - p') \mid k' \in \{1, 2\}, p' \in \mathcal{P}_{p-p_u}\},\\
&F((\text{inout}, L, R, s_u, p_u), 2, p) = s_u + \min\{F(L, 3, p') + F(R, 3, p - p'),\\
&\qquad\qquad F(L, 0, p') + F(R, 2, p - p') \mid p' \in \mathcal{P}_{p-p_u}\}.
\end{aligned}
$$

Note that the use of $F(R, 3, p - p')$ in the out- in inout-rules can avoid reaching the terminal condition $F((), 2, p) = \infty$. Thus, at least one neighbor has to be chosen to obtain a finite value of F.

Finally, we have to deal with "semi-solutions", i.e., the case $k = 3$:

$$
\begin{aligned}
&F((), 3, p) = \begin{cases} 0 : p = 0 \\ \infty : p > 0, \end{cases} \quad F((\text{leaf}, (), (), s_u, p_u), 3, p) = \begin{cases} s_u : p = p_u \\ \infty : p \neq p_u, \end{cases}\\
&F((\text{in}, L, R, s_u, p_u), 3, p) = s_u + \min\{F(L, k', p') + F(R, 3, p - p') \mid k' \in \{0, 3\}, p' \in \mathcal{P}_{p-p_u}\},\\
&F((\text{out}, L, R, s_u, p_u), 3, p) = s_u + \min\{F(L, k', p') + F(R, 3, p - p') \mid k' \in \{0, 1, 2\}, p' \in \mathcal{P}_{p-p_u}\},\\
&F((\text{inout}, L, R, s_u, p_u), 3, p) = s_u + \min\{F(L, k', p') + F(R, 3, p - p') \mid k' \in \{0, 3\}, p' \in \mathcal{P}_{p-p_u}\}.
\end{aligned}
$$

The structure $X(r)$ can be obtained in $\mathcal{O}(n)$ time from T and is recursively composed from $\mathcal{O}(n)$ structures belonging to leaves and edges of T. Values of F can be computed for $p \in \{0, \ldots, P\}$ from interior structures to larger composed structures, i.e., from the leaves to the root, by iterating through $\mathcal{P}_p$ or $\mathcal{P}_{p-p_u}$. Thus, the runtime to compute

$$\text{OPT}(I) = \max\{p \in \{0, \ldots, P\} \mid \exists_{k \in \{0,1,2\}} F(X(r), p, k) \leq c\}$$

is in $\mathcal{O}(n(P + 1)^2) \subseteq \mathcal{O}\left(nP^2 + n\right)$. This bound also covers the computation of P and the sets of neighbor vertices in $\mathcal{O}(n)$.

With $P = n$, the bound $\mathcal{O}\left(n^3\right)$ follows for uniform KP1N. □

6 Conclusions and outlook

Undirected co-graphs are well studied (Corneil et al. 1981) and can be interpreted as directed co-graphs by replacing each undirected edge with two bidirectional directed edges. It can be represented by a di-co-tree consisting of leaves and interior vertices for the "$\oplus$" and "$\otimes$" operations. The "$\oslash$" operation is not needed. Thus, upper runtime

bounds for uniform and general KPaN and KP1N problems on directed co-graphs also apply to undirected co-graphs. However, Proposition 1 provides the better runtime bound $\mathcal{O}(t \cdot P + n + m) \subseteq \mathcal{O}(n \cdot P + n^2)$ for the general all-neighbors problem on (undirected) graphs with n vertices, m edges and $t \leq n$ components, especially for co-graphs.

Given an undirected graph $G = (V, E)$, a set of vertices $V_0 \subseteq V$ fulfilling the one-neighbor condition on G is called a *one-neighbor set*. Lemma 5 in Borradaile et al. (2012) shows that by cutting a spanning tree of each connected component into small pieces, V can be partitioned into one-neighbor sets such that their induced graphs are stars. A *star* is a graph with a center vertex such that all edges are incident to the center vertex. The iterated disjoint union G' of these stars does not contain the induced graph P_4 and is therefore an undirected co-graph, cf. (Corneil et al. 1981). Each feasible solution of a KP1N problem on G' is also a feasible solution of the KP1N problem on G (but not necessarily vice versa). Thus, co-graphs can serve to find good but not necessarily optimal feasible KP1N solutions on undirected graphs.

The given pseudo-polynomial solutions can be used to obtain fully polynomial approximation schemes (FPTAS). One idea is to use a powerful result of Pruhs and Woeginger (2007) on the existence of an FPTAS for *subset selection problems*, which can be defined as follows. Given is a set $X = \{x_1, \ldots, x_n\}$ of n elements such that every element x_j has a positive profit p_j, and for every subset $X' \subseteq X$ it can be decided, based on a structure with l bits, in time polynomial in n and l, whether X' is a feasible solution. Further, there has to be a feasible solution for every instance. The task is to find a feasible solution X' realizing maximum profit. In Pruhs and Woeginger (2007) it has been shown that the existence of an algorithm for a subset selection problem with runtime polynomial in n, l, and $\sum_{i=1}^{n} p_i$ implies that there exists an FPTAS for this problem. In order to apply this result we have to know that knapsack with one-neighbor constraint and knapsack with all-neighbors constraint are subset selection problems. All properties are easy to verify. For the existence of a feasible solution we can choose the empty solution. The structure consists of the graph, the sizes, and the capacity. The graph can be represented by an adjacency matrix with $\mathcal{O}(n^2)$ bits, such that l is polynomially bounded by the input size.

However, we have to exclude zero profits in order to apply the result from Pruhs and Woeginger (2007). But for knapsack problems without zero profits, the stated upper term bounds apply all the more such that an FPTAS exists.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Bang-Jensen J, Gutin G (2009) Theory algorithms and applications. Springer, Berlin

Bang-Jensen J, Gutin G (eds) (2018) Classes of directed graphs. Springer, Berlin

Bang-Jensen J, Maddaloni A (2014) Arc-disjoint paths in decomposable digraphs. J. Graph Theory 77:89–110

Borradaile G, Heeringa B, Wilfong G (2011) The 1-neighbour knapsack problem. Springer, Berlin

Borradaile G, Heeringa B, Wilfong G (2012) The knapsack problem with neighbour constraints. J Discrete Algorithms 16:224–235

Corneil D, Lerchs H, Stewart-Burlingham L (1981) Complement reducible graphs. Discrete Appl Math 3:163–174

Crespelle C, Paul C (2006) Fully dynamic recognition algorithm and certificate for directed cographs. Discrete Appl Math 154(12):1722–1741

Goebbels StJ, Gurski F, Komander D (2021) The knapsack problem with special neighbor constraints on directed co-graphs. In: Proceedings of the international conference on operations research (OR 2021), Selected Papers of the International Annual Conference of the German Operations Research Society (GOR), Springer Verlag, to appear

Gourvès L, Monnot J, Tlilane L (2018) Subset sum problems with digraph constraints. J Comb Optim 36(3):937–964

Gurski F (2017) Dynamic programming algorithms on directed cographs. Stat Optim Inform Comput 5:35–44

Gurski F, Hoffmann S, Komander D, Rehs C, Rethmann J, Wanke E (2020) Computing directed steiner path covers for directed co-graphs. Springer, Berlin, pp 556–565

Gursk F, Komander D, Lindemann M (2020) Oriented coloring of msp-digraphs and oriented co-graphs. Springer, Berlin

Gurski F, Komander D, Lindemann M (2021) Homomorphisms to digraphs with large girth and oriented colorings of minimal series-parallel digraphs. Springer, Berlin

Gurski F, Komander D, Rehs C (2019) Computing digraph width measures on directed co-graphs. Springer, Berlin, pp 292–305

Gurski F, Komander D, Rehs C (2019) Oriented coloring on recursively defined digraphs. Algorithms 12(4):87

Gurski F, Komander D, Rehs C (2020) Solutions for subset sum problems with special digraph constraints. Math Methods Oper Res 92(2):401–433

Gurski F, Komander D, Rehs C (2020) Subset sum problems with special digraph constraints. In: Operations research proceedings (OR 2019), selected papers. Springer, pp 339–346 (2020)

Gurski F, Komander D, Rehs C (2021) How to compute digraph width measures on directed co-graphs. Theor Comput Sci 855:161–185

Gurski F, Rehs C (2018) Directed path-width and directed tree-width of directed co-graphs. Springer, Berlin, pp 255–267

Hellmuth M, Stadler P, Wieseke N (2017) The mathematics of xenology: di-cographs, symbolic ultrametrics, 2-structures and tree-representable systems of binary relations. J Math Biol 75(1):199–237

Johnson D, Niemi K (1983) On knapsacks, partitions, and a new dynamic programming technique for trees. Math Oper Res 8(1):1–14

Kellerer H, Pferschy U, Pisinger D (2010) Knapsack problems. Springer, Berlin

Kolliopoulos S, Steiner G (2007) Partially ordered knapsack and applications to scheduling. Discrete Appl Math 155(8):889–897

Nojgaard N, El-Mabrouk N, Merkle D, Wieseke N, Hellmuth M (2018) Partial homology relations - satisfiability in terms of di-cographs. Springer, Berlin, pp 403–415

Pferschy U, Schauer J (2017) Approximation of knapsack problems with conflict and forcing graphs. J Comb Optim 33:1300–1323

Pruhs K, Woeginger G (2007) Approximation schemes for a class of subset selection problems. Theor Comput Sci 382(2):151–156

Retoré C (1998) Pomset logic as a calculus of directed cographs. In: Proceedings of the fourth roma workshop: dynamic perspectives in logic and linguistics. CLUEB, pp 221–247

Valdes J, Tarjan R, Lawler E (1982) The recognition of series-parallel digraphs. SIAM J Comput 11:298–313