Solving the Rooted, Capacitated, Maximum Weight Connected Subgraph Problem on Graphs with Bounded Treewidth

William Surau
Ralf Borndörfer
surau@zib.de
borndoerfer@zib.de
Freie Universität Berlin & Zuse Institute Berlin
Berlin, Germany

Abstract

The Rooted Capacitated Maximum Weight Connected Subgraph Problem (RCMWCS) is to find a connected subgraph containing a prescribed root, subject to a global capacity constraint, that maximizes total profit. It generalizes several classical combinatorial optimization problems and is NP-hard even on trees. Building on a pseudo-polynomial-time dynamic programming algorithm for the RCMWCS on trees with integral capacities, based on the Left–Right method, we develop a fixed-parameter tractable algorithm for graphs of bounded treewidth. Our approach uses a dynamic program over a nice tree decomposition that explicitly tracks connectivity via partitions of bag vertices and respects capacity constraints. The resulting algorithm runs in time $k^{O(k)} \cdot C^2 \cdot |V|$, where k is the treewidth, $|V|$ the number of vertices of the graph, and C an upper bound on the capacity. This establishes fixed-parameter tractability of the RCMWCS with integral capacities parameterized by treewidth and the capacity bound C.

1 Introduction

Connected subgraph selection problems arise in many areas of combinatorial optimization, including network design, bioinformatics, and transportation planning. A prominent example is the *Maximum Weight Connected Subgraph* (MWCS) problem, which asks for a connected vertex-induced subgraph of maximum total profit. In many applications, however, solutions are additionally subject to global resource constraints (e.g., a budget or capacity limit) and must contain a designated root vertex. This motivates the study of the *Rooted Capacitated Maximum Weight Connected Subgraph Problem* (RCMWCS).

In the RCMWCS, each vertex is associated with a profit and a capacity consumption, and the objective is to find a connected subgraph containing a given root whose total capacity does not exceed a prescribed bound while maximizing profit. The problem generalizes classical problems such as Knapsack and is closely related in spirit to Steiner-type network design problems.

Related work. Rooted and constrained variants of the MWCS appear naturally as subproblems in applications. For example, Borndörfer et al. [5] study a rooted MWCS with lower/upper capacity bounds and additional balancing constraints in a district planning context, and identify rooted and budgeted MWCS-type subproblems as essential algorithmic building blocks in a column-generation approach to covering vertices by connected subgraphs under weight bounds [6]. A broad and systematic survey on graph covering and districting problems with connected subgraphs, including the role of constrained connected subgraph *pricing* problems (often taking the form of rooted and budgeted MWCS variants), is given by [14].

Budgeted versions of node-weighted connected subgraph problems have also been studied in their own right, in particular in conservation and corridor-design applications. A representative early model is the connected subgraph formulation of [7], which views the corridor as a connected set of parcels under a global acquisition budget. Building on this line, Dilkina and Gomes [9] investigate several mixed-integer programming formulations for enforcing connectivity under a budget constraint and compare flow-based and cut-based encodings.

For the rooted setting, [1] studies the Rooted Maximum Node-Weight Connected Subgraph problem and its budget-constrained variant (B-RMWCS). Their approach is again mixed-integer programming: connectivity is enforced via standard connectivity constraints (with separation in a branch-and-cut algorithm), while the budget dimension is handled by knapsack-type restrictions (strengthened by additional valid inequalities). Complementary to these exact MIP-based approaches, [2] proposes a relax-and-cut framework for large-scale constrained MWCS instances, combining Lagrangian relaxation with iterative cut generation and primal heuristics.

A powerful framework for exploiting structural properties of graphs is provided by the notion of treewidth. Many NP-hard problems admit fixed-parameter tractable algorithms when parameterized by treewidth, typically via dynamic programming on tree decompositions. While related Steiner-type problems have been extensively studied in this setting, see, e.g., [8], and the broader parameterized complexity of Steiner Tree variants has been investigated in detail, including directed settings [10], explicit treewidth parameterized treatments of rooted MWCS with global capacity constraints do not appear to exist in the literature. To the best of our knowledge, this paper is the first to contribute such a result.

In this paper, we first present a pseudo-polynomial-time dynamic programming algorithm for RCMWCS on trees with integral capacities, based on the Left–Right method. We then extend the study to graphs of bounded treewidth by developing a dynamic program over a nice tree decomposition. Our formulation explicitly tracks connectivity through partitions of bag vertices and incorporates capacity constraints into the state space.

Our contributions.

- We show that RCMWCS is NP-hard even on trees.
- We propose a pseudo-polynomial-time dynamic programming algorithm for trees with integral capacities.
- We develop a treewidth-based dynamic program for the RCMWCS with integral capacities on graphs of bounded

INOC '26, Liège (Belgium)

10.48786/inoc.2026.22

treewidth running in time $k^{O(k)} \cdot |V| \cdot C^2$, that is, we establish fixed-parameter tractability with respect to treewidth k and capacity bound C.

The remainder of the paper is organized as follows. In Section 2 we formally define the problem and discuss its complexity. Section 3 presents the dynamic programming algorithm for trees and its extension to graphs of bounded treewidth. We conclude with a discussion of the running time and practical limitations of the approach.

2 Problem Definition

For the *Rooted Capacitated Maximum Weight Connected Subgraph Problem (RCMWCS)*, we consider a graph $G = (V, E)$ with node capacities $c \in \mathbb{R}^V_{\geq 0}$ and profits $p \in \mathbb{R}^V$, as well as an upper capacity bound $C \in \mathbb{R}_{\geq 0}$, and a root node $r \in V$. For any subset of vertices $S \subseteq V$, we define

$$c(S) := \sum_{v \in S} c_v$$

and $p(S)$ analogously.

The RCMWCS is to find in G a tree $H = (V_H, E_H)$, such that $r \in V_H$ and $c(V_H) \leq C$, while $p(V_H)$ is maximized.

Since profits and capacities are associated with vertices only, it is without loss of generality to restrict attention to solutions that are trees.

Complexity. We show that RCMWCS is NP-hard by a reduction from Knapsack.

Consider an instance of Knapsack consisting of a set of $n \in \mathbb{N}$ items with weights $w \in \mathbb{Q}^n_{\geq 0}$, values $v \in \mathbb{Q}^n_{\geq 0}$, and a weight bound $B \in \mathbb{Q}$. We construct a star graph with n leaves and a center vertex serving as the root. For each leaf i, we set its capacity to $c_i = w_i$ and its profit to $p_i = v_i$. The root has capacity and profit equal to 0. The capacity bound is set to $C = B$. This construction is polynomial.

Any feasible solution to the resulting RCMWCS instance corresponds to selecting in addition to the root a subset of leaves whose total capacity does not exceed B, and whose total profit equals the sum of the selected values. Conversely, any feasible subset in the Knapsack instance yields a feasible connected subgraph containing the root. Hence, RCMWCS is NP-hard even on trees.

3 RCMWCS on Graphs with Bounded Treewidth

In this section, we discuss how to solve the RCMWCS on graphs with bounded treewidth. In the first part of this section, we propose a dynamic program (DP) for trees with integer capacities. Our algorithm runs in pseudo-polynomial time and uses the so-called Left-Right Method from Johnson et al. [12]. Following the discussion on trees, we develop a parameterized algorithm for graphs with bounded treewidth. In the remainder of the paper, we assume integral capacities and an integral capacity bound, which is necessary for our pseudo-polynomial dynamic programs.

3.1 Trees

The Left-Right Method, as introduced by Johnson et al. [12], traverses a tree in a Depth-First-Search (DFS) manner. This avoids the expensive merging of multiple children and their solutions, which is a big advantage over naive approaches. We apply it to a tree $T = (V, E)$ with root $r \in V$, node capacities $c \in \mathbb{N}^V_{\geq 0}$,

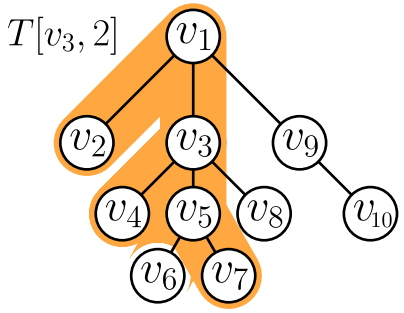

Figure 1: The orange subtree $T[v_3, 2]$ of the above DFS-ordered tree contains all vertices v_k with $k \leq 3$ and the first 2 children of v_3, together with all their successors.

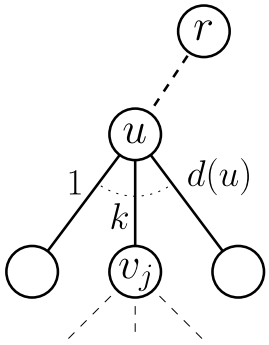

Figure 2: v_j is the k-th child of u.

profits $p \in \mathbb{R}^V$, and an upper capacity bound $C \in \mathbb{N}_{\geq 0}$. Let $v_1, \ldots, v_n$ be a DFS ordering of T with $v_1 = r$. Let $j \in \{1, \ldots, n\}$ and $i \in \{0, 1, \ldots, d(v_j)\}$, where $d(v_j)$ is the number of children of v_j. We denote by $T[v_j, i] \subseteq T$ the subtree containing all v_k with $k \leq j$, together with the first i children of v_j (in order of the indices) and all their successors, see Figure 1. We consider the following subproblem: among all subtrees $S \subseteq T[v_j, i]$ with $r, v_j \in V(S)$ and $c(V(S)) = \gamma$, maximize $p(V(S))$. We denote its optimal value by $dp[v_j, i, \gamma]$. It can be solved in a Left-Right DFS order, starting at the root node r with $i = 0$ and successively increasing i up to $d(r)$. Whenever the algorithm proceeds to the i-th child, the procedure is applied recursively.

More precisely, the base case is initialized for $\gamma \in \{0, ..., C\}$ as

$$dp[r, 0, \gamma] = \begin{cases} p(r), & \text{if } \gamma = c(r) \\ -\infty, & \text{otherwise.} \end{cases}$$

If $i = 0$ and v_j is the k-th child of u, see Figure 2, we compute

$$dp[v_j, 0, \gamma] = dp[u,\ k-1,\ \gamma - c(v_j)] + p(v_j).$$

Indeed, since no successor of v_j is considered, the value of $dp[v_j, 0, \gamma]$ is given by the profit $p(v_j)$ plus the optimal solution at the parent node using capacity $\gamma - c(v_j)$ and considering only the first $k-1$ children (recall that $v_j \in V(S)$).

For the case $1 \leq i \leq d(v_j)$, let w denote the i-th child of v_j. Then the recurrence is

$$dp[v_j, i, \gamma] = \max\Big\{dp[v_j, i-1, \gamma],\ dp[w, d(w), \gamma]\Big\}.$$

That is, we either exclude the i-th child w (and all of its descendants) from the solution, or we include w and take an optimal solution for the entire subtree $T[w, d(w)]$. Note that $T[w, d(w)]$

already contains all previously processed parts of the DFS traversal; hence no additional knapsack merge is needed.

The number of subproblems is $O(|V| \cdot C)$, and each recurrence can be computed in constant time. Overall, this yields a total running time of $O(|V| \cdot C)$. The optimal solution value is obtained as

$$\max_{\gamma \le C} dp[r, d(r), \gamma].$$

3.2 Bounded Treewidth

While the dynamic program from the previous section is restricted to trees, we now turn to graphs of bounded treewidth k. We briefly recall the notions of treewidth and tree decompositions and then present a fixed-parameter algorithm that runs in time $k^{O(k)} \cdot C^2 \cdot |V|$. Our approach follows the standard dynamic programming paradigm on nice tree decompositions and is inspired by treewidth-based algorithms for Steiner Tree (see, e.g., [8]). We extend these ideas to RCMWCS by incorporating a global capacity dimension.

Tree decompositions. A tree decomposition of a graph $G = (V, E)$ is a pair

$$\mathcal{T} = \big(T, \{X_t\}_{t \in V(T)}\big),$$

where T is a tree and each $X_t \subseteq V$ is a bag. To avoid ambiguities, we refer to elements of $V(T)$ as nodes and to elements of $V(G)$ as vertices. For a vertex $v \in V$, denote by T_v the subgraph of T induced by all nodes t with $v \in X_t$. Then $\mathcal{T}$ is a tree decomposition if it has the following properties:

(1) $\bigcup_{t \in V(T)} X_t = V$,

(2) $\forall \{u, v\} \in E : \exists t \in V(T) : \{u, v\} \subseteq X_t$,

(3) $\forall v \in V :\ T_v$ is connected.

The width of $\mathcal{T}$ is $\max_{t \in V(T)} |X_t| - 1$, and the treewidth of G is the minimum width over all tree decompositions of G.

Intuitively, each bag acts as a separator: removing X_t from G breaks the graph into components, and each such component is contained in the union of the bags of one connected component of the decomposition tree $T - t$. This separation property is the basis for dynamic programming on graphs of bounded treewidth.

Nice tree decompositions. To describe the dynamic program and to facilitate the analysis of its running time, we employ the concept of nice tree decompositions [4]. A nice tree decomposition is a normalized form of a tree decomposition that restricts the structure of the decomposition tree and the allowed changes between adjacent bags.

In our setting, a nice tree decomposition consists of the following node types: leaf nodes, introduce vertex nodes, forget nodes, join nodes, and introduce edge nodes. The bag of a leaf node contains exactly one vertex. We assume that there exists a leaf whose bag consists solely of the root vertex r, and we choose this node as the root of the decomposition tree. This induces a natural parent–child relation on the nodes of T.

Every introduce and forget node has exactly one child, whereas every join node has exactly two children. Let $t \in V(T)$ be a node and let t' denote its (unique) child, unless stated otherwise.

Introduce Vertex Node. The node t is an introduce vertex node if there exists a vertex $v \in V$ such that

$$X_t = X_{t'} \cup \{v\}.$$

Forget Node: The node t is a forget node if there exists a vertex $v \in V$ such that

$$X_t = X_{t'} \setminus \{v\}.$$

Join Node. The node t is a join node if it has exactly two children t_1 and t_2 with

$$X_t = X_{t_1} = X_{t_2}.$$

Introduce Edge Node. The node t is an introduce edge node if it is labeled with an edge $uv \in E$, and $u, v \in X_t$, and $X_t = X_{t'}$.

We schedule edge introductions as follows. Let t be a forget node that removes a vertex v, i.e., $X_t = X_{t'} \setminus \{v\}$. Consider the set of edges $vw \in E$ with $w \in X_t$ (equivalently, both endpoints v and w are contained in the bag $X_{t'}$ right before v is forgotten). We insert a (possibly empty) chain of introduce edge nodes between t' and t, one for each such edge vw, where each node keeps the same bag and is labeled with that edge. In this way, every edge is introduced when the first of its endpoints is forgotten; hence it is introduced exactly once.

Given a tree decomposition of G with n nodes, we can construct a nice tree decomposition with equal width and at most $4n$ nodes in $O(n)$ time [4].

Dynamic programming formulation. Consider an instance of RCMWCS with integral capacities together with a nice tree decomposition

$$\mathcal{T} = \big(T, \{X_t\}_{t \in V(T)}\big)$$

of G with width k. We assume that the decomposition is constructed such that there is a leaf node whose bag consists of the root vertex r only, and this node is chosen as the root of T.

For a node $t \in V(T)$, let G_t denote the subgraph of G whose vertex set consists of all vertices that appear in the bags of t and its descendants, together with all edges introduced in this part of the decomposition. For a subset $X \subseteq X_t$, a partition $\mathcal{P}$ of X, and a capacity value $\gamma \in \{0, \dots, C\}$, let $\mathrm{DP}[t, X, \mathcal{P}, \gamma]$ denote the maximum total profit of a subgraph S of G_t such that:

$$\mathrm{DP}[t, X, \mathcal{P}, \gamma] = \max_{S \subseteq G_t} p(S)$$

such that (1) $V(S) \cap X_t = X$,

(2) $\forall$ connected components K in $S : V(K) \cap X_t \in \mathcal{P}$,

(3) $\forall P \in \mathcal{P} : \exists$ connected component K in S : $V(K) \cap X_t = P$,

(4) $c(S) = \gamma$.

If no such set S exists, we set $\mathrm{DP}[t, X, \mathcal{P}, \gamma] := -\infty$.

In other words, we seek a subgraph of G_t with maximum total profit such that each connected component intersects the bag X_t in exactly one element of the partition $\mathcal{P}$; we refer to an element of $\mathcal{P}$ as *blocks*. Moreover, every block of $\mathcal{P}$ arises as the intersection of X_t with some connected component. Finally, the total capacity of the chosen subgraph is required to be exactly γ. Figure 3 illustrates the structure of such a subproblem together with a feasible solution.

Note that we do not require a partial solution in G_t to be connected to the root r, since in general $r \notin X_t$. The partition $\mathcal{P}$ only records how the connected components of the partial solution intersect the current bag X_t. Rooted connectivity is enforced only in the final state at the root bag $\{r\}$.

We now describe how the dynamic program is evaluated for each of the different node types.

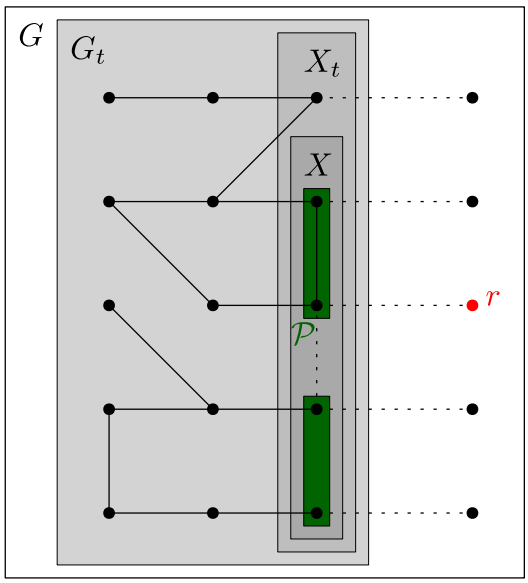

Figure 3: Dynamic programming problem at node $t \in V(T)$ with $X \subseteq X_t$ and partition $\mathcal{P}$ of X (green). The dotted edges indicate not yet introduced edges.

Leaf Node. Let $t \in V(T)$ be a leaf node and $v \in V$ with $X_t = \{v\}$. In case $X = \emptyset$, we initialize

$$\mathrm{DP}[t, \emptyset, \emptyset, 0] = 0,$$

and for $X = \{v\}$, we set

$$\mathrm{DP}[t, \{v\}, \{\{v\}\}, \gamma] = \begin{cases} p_v, & \text{if } \gamma = c_v \\ -\infty, & \text{otherwise.} \end{cases}$$

Introduce Vertex Node. Let $v \in V$ be the introduced vertex. In case $v \notin X$, we can retrieve the solution from the child node t' of t:

$$\mathrm{DP}[t, X, \mathcal{P}, \gamma] = \mathrm{DP}[t', X, \mathcal{P}, \gamma].$$

If $v \in X$, it has to be in an isolated block of $\mathcal{P}$, i.e., $\{v\} \in \mathcal{P}$, since we did not introduce any edge incident to v yet. Therefore, we get

$$\mathrm{DP}[t, X, \mathcal{P}, \gamma] = \begin{cases} \mathrm{DP}[t', X - v, \mathcal{P} - \{v\}, \gamma - c_v] + p_v, & \text{if } \{v\} \in \mathcal{P} \\ -\infty, & \text{otherwise.} \end{cases}$$

Introduce Edge Node. Let $uv \in E$ be the introduced edge, thus $u, v \in X_t$. Note that, introducing an edge does not force the edge to be part of any solution. Let $w \in X$, we denote the block in $\mathcal{P}$ containing w by $\mathcal{P}_w$.

We distinguish two cases depending on whether u and v lie in the same block of $\mathcal{P}$ or not.

If not, then either at least one of u and v is not contained in X, or they belong to two different blocks of $\mathcal{P}$. In either case, we can simply copy the value from the child node t'

$$\mathrm{DP}[t, X, \mathcal{P}, \gamma] = \mathrm{DP}[t', X, \mathcal{P}, \gamma].$$

For the case u and v are in the same block $\mathcal{P}_u = \mathcal{P}_v$, we can obtain the solution from t' by going through all its partitions and combine the blocks of u and v. For any partition $\mathcal{P}'$ of X such that:

$$\mathcal{P}' - (\mathcal{P}'_u \cup \mathcal{P}'_v) = \mathcal{P} - \mathcal{P}_v, \tag{1}$$

the solution of $\mathrm{DP}[t', X, \mathcal{P}', \gamma]$ is feasible for $\mathrm{DP}[t, X, \mathcal{P}, \gamma]$. For $\mathcal{P}' = \mathcal{P}$, this is immediate. Otherwise, choosing the edge uv does not change the objective value, and it connects the connected components corresponding to $\mathcal{P}'_u$ and $\mathcal{P}'_v$.

The solution of $\mathrm{DP}[t, X, \mathcal{P}, \gamma]$ can be obtained by considering all partitions $\mathcal{P}'$ of X that satisfy (1):

$$\mathrm{DP}[t, X, \mathcal{P}, \gamma] = \max_{\substack{\mathcal{P}' \text{ of } X: \\ \mathcal{P}' \text{ satisfies (1)}}} \mathrm{DP}[t', X, \mathcal{P}', \gamma].$$

This yields the optimal value. Let G^* be the optimal subgraph from the solution of $\mathrm{DP}[t, X, \mathcal{P}, \gamma]$ and let $\mathcal{P}'$ of X be the partition corresponding to the connected components of $G^* - uv$. If

$$\mathrm{DP}[t', X, \mathcal{P}', \gamma] < \mathrm{DP}[t, X, \mathcal{P}, \gamma],$$

we found a better solution for $\mathrm{DP}[t', X, \mathcal{P}', \gamma]$ and hence it was not optimal.

Forget Node. Let $v \in V$ be the vertex that is forgotten at node t. There are two possibilities: either v is contained in the optimal subgraph or it is not. If v is included, the solution can be obtained from the child node t' by assigning v to one of the blocks of the partition. Among all such possibilities, we select the one that yields the maximum profit. Formally, let $\mathcal{P}_{\max}$ be a partition that satisfies

$$\mathcal{P}_{\max} := \operatorname*{arg\,max}_{\substack{\mathcal{P}' \text{ of } X+v: \\ \mathcal{P}' \cap \mathcal{P} = \mathcal{P}' \setminus \mathcal{P}'_v}} \mathrm{DP}[t', X + v, \mathcal{P}', \gamma].$$

If v is not part of the solution, we simply inherit the value $\mathrm{DP}[t', X, \mathcal{P}, \gamma]$ from the child. Taking the better of the two cases yields

$$\mathrm{DP}[t, X, \mathcal{P}, \gamma] = \max\left\{\mathrm{DP}[t', X + v, \mathcal{P}_{\max}, \gamma], \mathrm{DP}[t', X, \mathcal{P}, \gamma]\right\}.$$

Join Node. Let $t_1, t_2 \in V(T)$ be the two children of a join node t, with identical bags $X_t = X_{t_1} = X_{t_2}$. To compute the value $\mathrm{DP}[t, X, \mathcal{P}, \gamma]$, we combine the solutions of the two child nodes.

To this end, we first define how to merge two partitions $\mathcal{P}_1$ and $\mathcal{P}_2$ of X, denoted by $\mathcal{P}_1 \sqcup \mathcal{P}_2$. We construct an auxiliary graph U whose nodes correspond to the blocks of $\mathcal{P}_1$ and $\mathcal{P}_2$. Two nodes in U are connected by an edge if their corresponding vertex sets intersect. Each connected component of U induces a block consisting of the union of its vertex sets. The collection of these unions defines a new partition.

The dynamic programming value is then given by

$$\mathrm{DP}[t, X, \mathcal{P}, \gamma] = \max_{\substack{\mathcal{P}_1 \sqcup \mathcal{P}_2 = \mathcal{P} \\ \gamma_1 + \gamma_2 - c(X) = \gamma}} \Big(\mathrm{DP}[t_1, X, \mathcal{P}_1, \gamma_1] + \mathrm{DP}[t_2, X, \mathcal{P}_2, \gamma_2] - p(X)\Big).$$

We have described how the parameterized algorithm processes each type of node in the decomposition. The dynamic program is evaluated in a bottom–up fashion, that is, from the leaf nodes towards the root of the decomposition tree. The optimal solution value is then obtained as

$$\max_{\gamma \le C} \mathrm{DP}[r_T, \{r\}, \{\{r\}\}, \gamma].$$

We now analyze the running time of this dynamic program. The number of distinct partitions of a set of size m is given by the Bell number B_m [3]. Moreover, a simple upper bound is $B_m \le m^m$.

Summing over all subsets $X \subseteq X_t$ with $|X_t| \leq k+1$, we use the classical Bell number recurrence

$$B_{k+2} = \sum_{i=0}^{k+1} \binom{k+1}{i} B_i$$

[3]. Therefore, the number of pairs $(X, \mathcal{P})$, where $X \subseteq X_t$ and $\mathcal{P}$ is a partition of X, is at most B_{k+2}, and thus bounded by $(k+2)^{k+2}$. Since for each such pair and each capacity value $\gamma \in \{0, \ldots, C\}$ the dynamic program stores one entry, the total number of states per decomposition node is bounded by

$$(k+2)^{k+2} \cdot (C+1) = k^{O(k)} \cdot C.$$

The running time is dominated by join nodes, since computing their DP values requires combining all pairs of states from the two child nodes. Joining two nodes leads to the following running time:

$$\left(k^{O(k)} \cdot C\right)^2 = k^{O(k)} \cdot C^2$$

Merging two partitions can be done with a union–find data structure and contributes only an additional factor of $k^{O(k)}$.

A nice tree decomposition with additional introduce edge nodes can be constructed with $O(k \cdot |V| + |E|)$ nodes [8]. Moreover, since graphs of treewidth k are partial k-trees [13], they satisfy $|E| = O(k \cdot |V|)$ [11]. Therefore, we conclude a running time of

$$k^{O(k)} \cdot C^2 \cdot O(k \cdot |V|) = k^{O(k)} \cdot C^2 \cdot |V|.$$

4 Conclusion

We studied the Rooted Capacitated Maximum Weight Connected Subgraph Problem and analyzed its algorithmic complexity under structural restrictions. After establishing NP-hardness even on trees, we presented a pseudo-polynomial-time dynamic programming algorithm for trees with integral capacities. Building on this result, we developed a fixed-parameter tractable algorithm for graphs of bounded treewidth using a dynamic program over a nice tree decomposition.

The resulting running time of $k^{O(k)} \cdot C^2 \cdot |V|$ shows that the RCMWCS with integral capacities is fixed-parameter tractable when parameterized by treewidth and the capacity bound. The algorithm extends existing techniques for Steiner-type problems to a significantly richer, capacity-constrained setting. Future work includes tightening the state space using refined connectivity encodings, improving the dependence on the capacity parameter, and investigating practical heuristics or approximations inspired by the presented dynamic programs.

References

[1] Eduardo Álvarez-Miranda, Ivana Ljubić, and Petra Mutzel. 2013. The Rooted Maximum Node-Weight Connected Subgraph Problem. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2013) (Lecture Notes in Computer Science, Vol. 7874)*. Springer, 300–315. doi:10.1007/978-3-642-38171-3_20

[2] Eduardo Álvarez-Miranda and Markus Sinnl. 2017. A Relax-and-Cut framework for large-scale maximum weight connected subgraph problems. *Computers & Operations Research* 87 (2017), 63–82. doi:10.1016/j.cor.2017.05.015

[3] Eric Temple Bell. 1934. Exponential Polynomials. *Annals of Mathematics* 35, 2 (1934), 258–277. doi:10.2307/1968431

[4] Hans L Bodlaender and Arie MCA Koster. 2008. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.* 51, 3 (2008), 255–269.

[5] Ralf Borndörfer, Stephan Schwartz, and William Surau. 2022. Rooted Maximum Weight Connected Subgraphs with Balancing and Capacity Constraints. In *Proceedings of the 10th International Network Optimization Conference (INOC), Aachen, Germany, June 7-10, 2022*. 63 – 68. doi:10.48786/inoc.2022.12

[6] Ralf Borndörfer, Stephan Schwartz, and William Surau. 2023. Vertex Covering with Capacitated Trees. *Networks* 81, 2 (2023), 253–277. doi:10.1002/net.22130

[7] Jon M. Conrad, Carla P. Gomes, Willem-Jan van Hoeve, Ashish Sabharwal, and Jordan F. Suter. 2012. Wildlife corridors as a connected subgraph problem. *Journal of Environmental Economics and Management* 63, 1 (2012), 1–18. doi:10.1016/j.jeem.2011.08.001

[8] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. 2015. *Parameterized algorithms*. Vol. 5. Springer. 151–175 pages.

[9] Bistra Dilkina and Carla P. Gomes. 2010. Solving Connected Subgraph Problems in Wildlife Conservation. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2010) (Lecture Notes in Computer Science, Vol. 6140)*. Springer, 102–116. doi:10.1007/978-3-642-13520-0_14

[10] Andreas Emil Feldmann and Dániel Marx. 2023. The Complexity Landscape of Fixed-Parameter Directed Steiner Network Problems. *ACM Transactions on Computation Theory* 15, 3-4 (2023). doi:10.1145/3580376

[11] Yuan Hong. 2004. Tree-width, clique-minors, and eigenvalues. *Discrete Mathematics* 274, 1 (2004), 281–287. doi:10.1016/S0012-365X(03)00199-7

[12] David S Johnson and KA Niemi. 1983. On knapsacks, partitions, and a new dynamic programming technique for trees. *Mathematics of Operations Research* 8, 1 (1983), 1–14.

[13] Jan V Leeuwen. 1990. *Handbook of theoretical computer science: Algorithms and complexity*. Mit Press.

[14] Stephan Schwartz. 2022. An overview of graph covering and partitioning. *Discrete Mathematics* 345, 8 (2022), 112884. doi:10.1016/j.disc.2022.112884