Algorithmica (2011) 60: 274–300
DOI 10.1007/s00453-009-9336-x

Capacitated Domination Problem

Mong-Jen Kao · Chung-Shou Liao · D.T. Lee

Received: 22 October 2008 / Accepted: 19 June 2009 / Published online: 9 July 2009

Abstract We consider a generalization of the well-known domination problem on graphs. The (*soft*) capacitated domination problem with demand constraints is to find a dominating set D of minimum cardinality satisfying both the capacity and demand constraints. The capacity constraint specifies that each vertex has a capacity that it can use to meet the demands of dominated vertices in its closed neighborhood, and the number of copies of each vertex allowed in D is unbounded. The demand constraint specifies the demand of each vertex in V to be met by the capacities of vertices in D dominating it. In this paper, we study the capacitated domination problem on trees from an algorithmic point of view. We present a linear time algorithm for the unsplittable demand model, and a pseudo-polynomial time algorithm for the splittable demand model. In addition, we show that the capacitated domination problem on trees with splittable demand constraints is NP-complete (even for its integer version) and provide a polynomial time approximation scheme (PTAS). We also give a primal-dual approximation algorithm for the weighted capacitated domination problem with splittable demand constraints on general graphs.

Keywords Algorithm · Domination · Facility location

An extended abstract of this paper appeared in the 18th International Symposium on Algorithms and Computation (ISAAC), pp. 256–267, 2007.

Supported in part by the National Science Council under the Grants NSC95-2221-E-001-016-MY3, NSC96-2752-E-002-005-PAE, NSC96-2217-E-001-001, NSC96-3114-P-001-002-Y, and NSC96-3114-P-001-007.

M.-J. Kao · C.-S. Liao (✉) · D.T. Lee
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan
e-mail: lcs.shou@gmail.com

C.-S. Liao · D.T. Lee
Institute of Information Science, Academia Sinica, Nankang, Taipei 115, Taiwan

1 Introduction

The domination problem on graphs is one of the well-known combinatorial optimization problems. The domination problem can be described as follows. Let $G = (V, E)$ denote an undirected graph with vertex set V and edge set E. G is a weighted graph if each vertex $v \in V$ is associated with a weight $w(v) \in \mathbb{R}^+$, where $\mathbb{R}^+$ denotes the set of non-negative real numbers. A vertex v is said to *dominate* itself and each of its neighbors. A set $D \subseteq V$ is called a *dominating set* if every vertex in V is dominated by at least one vertex in D. The goal is to find an optimal dominating set D^* of minimum weight in G. The weight of D^* is equal to the sum of the weights of all the vertices in D^*, and the minimum weight of D^* is called the *weighted domination number* of G, denoted $\gamma_w(G)$. If the weight of each vertex is unity, then $\gamma_w(G)$ would be the cardinality of the optimal dominating set D^*, and it is called the *domination number* of G, denoted $\gamma(G)$.

There has been considerable amount of research devoted to the metric case of the *capacitated facility location* problem. The capacitated facility location problem is defined as follows. Consider a set C of clients and a set F of facilities. Each client has associated with it a *demand* and each facility has a *capacity* that specifies the *maximum* service the facility can provide to its clients to meet their demands. In addition a facility also has a *setup* or *operating cost* if it is opened and *service* cost, which is based on a predefined function $\mathcal{A} : F \times C \to \mathbb{R}^+$, among pairs of facilities and clients, and $\mathcal{A}(f, c)$, $f \in F, c \in C$, denotes the service cost when client c is assigned to be serviced by facility f. The metric service function, or the metric distance function, $\mathcal{A}$, is nonnegative and symmetric, and obeys the triangle inequality. The goal is to find a subset $K \subseteq F$ of facilities to set up and an assignment of clients to facilities, such that the demand requirements of all the clients are satisfied, facilities capacities are not violated, and the total cost, including facility setup and service cost, is minimized. The capacity of a facility in the capacitated facility location problem is said to be *hard* if the facility can be opened at most a certain *limited* number of times to serve clients' demands; it is said to be *soft* if the facility can be opened without any restrictions on the number of times it can be opened. In addition, the demand of a client is called *unsplittable* if we require that the entire demand of the client is served by a *single* facility; otherwise, we call the demand *splittable*. In general, for each client v, let the *maximum* number of distinct facilities to serve a client[1] v be called the *demand split number* of v, denoted k_v. If $k_v = 1$ for each demand v, then we call this an *unsplittable* model; and if $k_v = \infty$, then we call this a *splittable* model. When $k_v = m$ for some integer m, then we call this an *m-splittable* model.

The *soft* capacitated facility location problem was considered in the literature [1, 10, 20, 21, 29, 31] and approximation algorithms based on some linear programming techniques, like LP-rounding and primal-dual algorithms were obtained. In 1997 Shmoys et al. [31] first presented a 5.69-approximation algorithm for the soft uniform capacity model with demand constraints. Chudak and Shmoys [10] improved this approximation factor to 3. Using primal-dual method Jain and Vazirani [20] gave a 4-approximation algorithm for the generalized nonuniform capacity model. Arya

[1] We shall without confusion use the terms, client and its demand, interchangeably.

et al. [1] improved this approximation factor to $2+\sqrt{3}$ based on local search heuristics. Following the method of Jain and Vazirani [20], Jain et al. [21] further improved the factor to 3. The latest result, with factor 2, due to Mahdian et al. [29], achieves the integrality gap of the natural LP relaxation of this soft capacity model. The *hard* capacitated facility location problem was treated differently however. Because of the large integrality gap, linear programming techniques do not work efficiently, and local search heuristics were proposed instead. Korupolu et al. [23] first gave a $(8+\epsilon)$-approximation algorithm for the hard uniform capacity model. Chudak and Williamson [11] improved the approximation factor to $(6+\epsilon)$. Based on the scaling technique, Charikar and Guha [5] improved the factor to $(3+2\sqrt{2})$. Levi et al. [24] further improved the factor to 5. Pál et al. [30] presented a $(9+\epsilon)$-approximation algorithm for the generalized hard nonuniform capacity model. This was improved by Mahdian and Pál. [28], and Zhang et al. [32] to yield an approximation factor $(3+2\sqrt{2})$. In addition, unlike the facility location problem, it is not known whether the capacitated k-median problem has a constant approximation [4–6, 8, 20, 23], even if the capacities are soft. There were only constant-factor ratio approximation algorithms which exceed either k [4, 23] or the capacity constraint [6, 8]. The difference between the capacitated k-median problem and the capacitated facility location problem is the extra constraint that the number of opening facilities is at most k.

In this paper we investigate the capacitated domination problem on graphs, which is closely related to the capacitated facility location problem, in which the clients and facilities are vertices of the graphs, and the set of facilities we open corresponds to the dominating set, and all the distances are either 0 or ∞ (note that this is not a metric distance function). The capacitated domination problem was introduced by Haynes et al. [17], who discussed a lot of variations of domination problem. In addition, the k-tuple domination problem, i.e., to find a minimum vertex subset such that every vertex in the graph is dominated by at least k vertices in this set, which was investigated by Liao and Chang [25, 26] in 2002, is slightly similar to the capacitated domination problem with k-splittable uniform demand model. They studied it on special graph classes from an algorithmic point of view. The concept of capacitated domination problem with demand constraints in graph theory is a more natural model for many facility location problems in practice. For example, consider the connected domination problem on unit-disk graphs [7], one of the virtual backbone models for an ad-hoc wireless network, in which the users or clients have demands, while each transmitter providing the services to users has a capacity.

The *capacitated vertex cover problem* is another famous problem related to the capacitated domination problem on graphs. Chuzhoy and Naor [9] proved weighted capacitated vertex cover problem with hard capacity and uniform demand is at least as hard as the *set cover* problem and provided a logarithmic approximation algorithm. For unweighted version with hard capacity and unsplittable nonuniform demand, they proved that this model is unapproximable unless P = NP and gave a 3-approximation algorithm for this model when each edge demand is uniform. Gandhi et al. [14] further improved the approximation ratio 3 of the latter model to 2, which is the best known ratio for the general vertex cover problem. In addition, Gandhi et al. [15] obtained a 2-approximation algorithm for the weighted version, which allows some vertices to exceed at most two times their capacity constraints. About the soft capacity model, Gandhi et al. [15] provided an LP-rounding 2-approximation algorithm for

the soft capacitated vertex cover problem with uniform demand. Guha et al. [16] presented the same result by a primal-dual method, and a 3-approximation algorithm for the generalized model with soft capacity and splittable nonuniform demand. They also gave polynomial time algorithms for some restricted cases of soft capacitated vertex cover problem on trees and showed that the weighted version on trees with splittable uniform demand is NP-hard.

Our Contribution To the best of our knowledge this is the first paper considering both notions of capacity and demand in domination problem on graphs, which is as hard as *set cover* problem (a reduction from set cover problem to domination problem on *split graphs* is intuitive). In particular, we study the *soft* capacitated domination problem with demand constraints on trees from an algorithmic point of view. We present a linear time algorithm for the *unsplittable* demand model. For the *splittable* demand model on trees, we show it is NP-complete even when the vertex capacity and demand are integers, in contrast to the linear time result of Guha et al. [16] for the same model of capacitated vertex cover on trees. Furthermore, based on our NP-completeness reduction, we develop a pseudo-polynomial time algorithm and further a combinatorial polynomial time approximation scheme (PTAS) for splittable demand model on trees. We also give a primal-dual $(\Delta + 1)$-factor approximation algorithm for weighted capacitated domination problem with splittable demand constraints on general graphs, where Δ is the maximum degree of the vertices. The approximation factor is almost equal to one of the two well-known approximation ratios (Δ [2, 18] and $O(\ln n)$ [12, 22, 27]) of general domination problem. We remark that Bar-Ilan et al. [3] considered the non-metric capacitated facility location problem (a generalization of capacitated domination problem) and presented an $O(\log n + \log M)$ approximation algorithm for the integer version, which is close to the threshold $O(\ln n)$ [13] for approximating domination problem, where M is the largest parameter.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. A vertex $w \in V$ is said to be *a neighbor of* or *adjacent to* a vertex $v \in V$ if $(v, w) \in E$. The *neighborhood* of a vertex $v \in V$ is $N_G(v) = \{w \in V \mid (v, w) \in E\}$. The *closed neighborhood* of $v \in V$ is $N_G[v] = N_G(v) \cup \{v\}$. The closed degree of a vertex $v \in V$ is $deg[v] = |N_G[v]|$. The closed degree of a graph G is $\Delta_G^* = \max_{v \in V} deg[v]$.

In the *capacitated domination problem with demand constraints*, each vertex $v \in V$ has a positive demand $d(v)$, required of service from vertices in $N_G[v]$, and a positive capacity $c(v)$, which is the *maximum* service v can provide to vertices in $N_G[v]$. A multi-set $D \subseteq V$, with a *multiplicity function* $x : V \to \mathbb{Z}^+ \cup \{0\}$, is called a *capacitated dominating multi-set* if there exists an assignment function $f : V \times D \to \mathbb{R}^+ \cup \{0\}$ such that for every $v \in V$ and $\delta \in D \cap V$, the following constraints $(*)$ are satisfied. Note that $f(v, \delta)$ denotes the amount of demand requirement of each vertex

$v \in V$ that is assigned to a vertex $\delta \in D \cap V$, and $f(v, \delta) = 0$ if $v \notin N_G[\delta]$.

$$\begin{aligned} &1. \quad \sum_{\forall \delta \in N_G[v] \cap D} f(v, \delta) \geq d(v) \quad \text{(demand constraint for } v\text{)}, \\ &2. \quad \sum_{\forall v \in N_G[\delta]} f(v, \delta) \leq x(\delta) \times c(\delta) \quad \text{(capacity constraint for } \delta\text{)}. \end{aligned} \tag{$*$}$$

The first constraint stipulates that the demand $d(v)$ for each vertex must be met by the service rendered by vertices in $N_G[v] \cap D$. The second constraint restricts that the service of any vertex $\delta \in V \cap D$ rendered to the vertices in $N_G[\delta]$ should not exceed its total available capacity, i.e., $x(\delta) \times c(\delta)$, where $x(\delta)$ denotes the number of copies of δ in D.

More formally, given a graph $G = (V, E)$, a capacity function $c : V \to \mathbb{R}^+ \cup \{0\}$, and a demand function $d : V \to \mathbb{R}^+ \cup \{0\}$, the *capacitated domination problem with demand constraints* is to find a *capacitated dominating multi-set* D of G and an assignment function $f : V \times D \to \mathbb{R}^+ \cup \{0\}$, which satisfies both the demand constraint and capacity constraint (specified in ($*$) such that $\sum_{v \in V} x(v)$ is minimum, where $x(v)$, defined by a multiplicity function $x : V \to \mathbb{Z}^+ \cup \{0\}$, denotes the number of copies of $v \in V$ that belongs to D. $\sum_{v \in V} x(v)$ is the *cardinality* or the *size* of the multi-set D, denoted $|D|$.

One can consider the weighted capacitated domination problem by associating with each vertex a positive real weight $w(v)$, which denotes the cost incurred and may differ for different vertex v. Then the problem becomes that of finding a capacitated dominating multi-set such that the total weight $\sum_{v \in V} w(v) \times x(v)$ is minimum. There are also variations in the assignment of demands for each vertex and in the selection of multiple copies of vertices. We say the demand is unsplittable if we require that $f(v, \delta)$ is either $d(v)$ or 0 for each $v \in V$ and $\delta \in N_G[v] \cap D$, and splittable if no such restriction exists on f. The capacity is said to be *soft* if the number of available copies of each vertex is unbounded and *hard* otherwise.

The rest of this paper is organized as follows. In Sect. 3, we consider the soft capacitated domination problem with demand constraints on trees. We give a linear time algorithm for unsplittable demand model. For splittable demand model, we show the NP-completeness result and present a pseudo-polynomial time algorithm. In Sect. 4, we provide a polynomial time approximation scheme for splittable demand model on trees. We also consider the weighted capacitated domination problem on general graphs, and provide a Δ^*-approximation algorithm for splittable demand model. Finally we conclude in Sect. 5 with some discussions of *hard* capacity model and future work.

3 Capacitated Domination on Trees

Given a tree $T = (V, E)$ with capacity function c and demand function d, we shall consider the capacitated domination problem with demand constraints on T. Guha et al. [16] showed in 2002, that the *weighted* capacitated vertex cover problem on trees is NP-hard and that the unweighted case can be solved in linear time. In this section,

however, we show that capacitated domination problem on trees for the unweighted case is NP-complete when the demand is splittable.

In Sect. 3.1, we show that capacitated domination problem with unsplittable demand on trees can be solved in linear time. In the later subsections, we consider the case when the vertex demand is splittable. As a contrast to the linear time result given by Guha et al. [16] in capacitated vertex cover problem on trees, we show in Sect. 3.2 that capacitated domination problem with splittable demand on trees is NP-complete, even when the vertex capacity and demand are integers. In Sect. 3.3, we present a pseudo-polynomial time algorithm for the integer version of the splittable demand model.

3.1 Unsplittable Demand Model

In this subsection we show that capacitated domination problem with unsplittable demand on trees can be solved in linear time. In particular, we consider a stronger version of the original problem to find an optimal capacitated dominating multi-set D with the maximum *residue capacity* at the root. The residue capacity of a vertex $\delta \in D$, denoted by $RC[\delta]$, is the remainder of its capacity after δ serves the demands of vertices in $N_G[\delta]$, i.e., $RC[\delta] = x(\delta) \times c(\delta) - \sum_{\forall v \in N_G[\delta]} f(v, \delta)$. The algorithm runs in a bottom-up manner. It processes one vertex at each iteration in the postorder tree traversal. Since the demand is unsplittable, we will use the words *assign the demand of v to δ* or *assign v to δ* alternatively for convenience.

Given a tree $T = (V, E)$ with a postorder tree traversal, let $p(v)$ be the parent of $v \in V$, $Ch(v)$ the child set of v, and T_v the subtree rooted at v. At each iteration i, the algorithm computes the minimum cardinality of the capacitated dominating multi-set D_i and the residue capacity of v_i for T_{v_i} when v_i assigned to itself or to one of its children in $Ch(v_i)$ or when v_i assigned to its parent $p(v_i)$. The minimum cardinality of the capacitated dominating multi-set D_i and the residue capacity of v_i in the former case are denoted $W_{\downarrow}[v_i]$ and $RC_{\downarrow}[v_i]$ respectively and in the latter they are denoted $W_{\uparrow}[v_i]$ and $RC_{\uparrow}[v_i]$ respectively. $W_{\uparrow}[v_i]$ is assumed to be infinity if v_i is the root of T. Note that in the former case when there is more than one child of v_i providing minimum $W_{\downarrow}[v_i]$, we select the one with the maximum $RC_{\downarrow}[v_i]$, and in case of a tie, we break it arbitrarily.

Lemma 3.1 *Given a postorder tree traversal $v_1, v_2, \ldots, v_n$, we can at each iteration i, $1 \le i \le n$, immediately determine the assignment of the demand of v_i optimally to its parent $p(v_i)$ or otherwise, except for the case when $W_{\downarrow}[v_i] = W_{\uparrow}[v_i]$ and $RC_{\downarrow}[v_i] > RC_{\uparrow}[v_i]$ (we call this case undetermined condition of v_i).*

Proof If $W_{\uparrow}[v_i] \neq W_{\downarrow}[v_i]$, we claim that the smaller one is the optimal demand assignment. Suppose $W_{\uparrow}[v_i] > W_{\downarrow}[v_i]$. If there were an optimal solution in which v_i is assigned to its parent $p(v_i)$, then we would simply use the demand assignment corresponding to $W_{\downarrow}[v_i]$ and put an additional copy at $p(v_i)$. This solution has either smaller cardinality, or the same cardinality but more residue capacity at $p(v_i)$. The case $W_{\uparrow}[v_i] < W_{\downarrow}[v_i]$ is obvious. As for $W_{\uparrow}[v_i] = W_{\downarrow}[v_i]$ and $RC_{\uparrow}[v_i] \ge RC_{\downarrow}[v_i]$, we would simply assign v_i, to its parent $p(v_i)$ to obtain a larger residue capacity at $p(v_i)$. □

The main idea of our algorithm, ALGORITHM MCDUT (see Appendix), is described in the following. Given a tree $T = (V, E)$ with a postorder tree traversal $v_1, v_2, \ldots, v_n$, we shall process vertices one at a time in the postorder. We maintain four variables, $W_\uparrow[v_i]$, $W_\downarrow[v_i]$, $RC_\uparrow[v_i]$, $RC_\downarrow[v_i]$, and one pointer from v_i to v_j, if necessary, where v_j is the child of v_i which gives the optimal $W_\downarrow[v_i]$, among all children of v_i. For convenience we assume that the capacity of the parent of v_i, $c(p(v_i))$ is available when we process v_i. At each iteration i, we do the following computations.

Case (1) Computation of $W_\uparrow[v_i]$ and $RC_\uparrow[v_i]$. If v_i is to be assigned to $p(v_i)$, we compute $W_\uparrow[v_i]$ as (1) the number of copies of $p(v_i)$ needed, i.e., $\lceil \frac{d(v_i)}{c(p(v_i))} \rceil$, plus (2) the summation of $\min\{W_\downarrow[u], W_\uparrow[u]\}$ for every child u of v_i, and minus (3) the saving of copies of v_i , i.e., $\lfloor \frac{RC[v_i]}{c(v_i)} \rfloor$, provided by the residue capacity of v_i due to the assignments of its children, if any, where $RC[v_i] = \sum_{u \in Ch(v_i), W_\downarrow[u] \geq W_\uparrow[u]} (c(v_i) - (d(u) \bmod c(v_i)))$.
We compute $RC_\uparrow[v_i]$ as $RC[v_i] \bmod c(v_i)$.

Case (2) Computation of $W_\downarrow[v_i]$ and $RC_\downarrow[v_i]$.

Case (2-1) If v_i is assigned to itself, we compute $W_\downarrow[v_i]$ in a way similar to Case (1) with some modifications: part (1) is modified to be the number of copies of v_i, i.e., $\lceil \frac{d(v_i)}{c(v_i)} \rceil$. The computation of part (2) is the same. The computation of part (3) the savings of copies of v_i, is the same as before, i.e., $\lfloor \frac{RC[v_i]}{c(v_i)} \rfloor$, except that $RC[v_i]$ given above needs to be adjusted by adding $(c(v_i) - (d(v_i) \bmod c(v_i)))$.

Case (2-2) If v_i is assigned to one of its children, say $u^* \in Ch(v_i)$, we compute $W_\downarrow[v_i]$ in a way similar to Case (1) with the following modifications: part (1) is modified to be the number of copies of u^*, i.e., $\lceil \frac{d(v_i)}{c(u^*)} \rceil$. The computation of part (2) is the same as above and the computation of part (3) is modified, depending on the status of u^*.

Case (2-2-1) Assignment of u^* is determined We add the saving of copies of u^*, $\lfloor (RC[u^*] + (c(u^*) - (d(v_i) \bmod c(u^*))))/c(u^*) \rfloor$, to part (3), where $RC[u^*] = RC_\uparrow[u^*]$ if u^* was assigned upward by Lemma 3.1; and $RC[u^*] = RC_\downarrow[u^*]$, if u^* was assigned downward.

Case (2-2-2) Assignment of u^* is undetermined In this case, $W_\uparrow[u^*] = W_\downarrow[u^*]$ and $RC_\uparrow[u^*] < RC_\downarrow[u^*]$. We need to consider both cases in which u^* is assigned upward and downward, in order to obtain the optimal $W_\downarrow[v_i]$. The case when u^* is assigned upward is the same as Case (2-2-1). For the case when u^* is assigned downward, we need to modify $RC[v_i]$ given in Case (1) by subtracting from it $(c(v_i) - (d(u^*) \bmod c(v_i)))$ and then compute for part (3) $\lfloor \frac{RC[v_i]}{c(v_i)} \rfloor + \lfloor (RC_\downarrow[u^*] + (c(u^*) - (d(v_i) \bmod c(u^*))))/c(u^*) \rfloor$.

We compute $RC_\downarrow[v_i]$ as $RC[v_i] \bmod c(v_i)$ for each subcase Case (2-1) and Case (2-2).

After we process Case (2-1) and Case (2-2), we would obtain $W_\downarrow[v_i]$ for the case, when v_i is assigned to itself and when v_i is assigned to each child $u^* \in Ch(v_i)$,

and obtain $RC_{\downarrow}[v_i] = RC[v_i] \bmod c(v_i)$ accordingly. We shall select as the optimal $W_{\downarrow}[v_i]$ the assignment for v_i that gives the minimum $W_{\downarrow}[v_i]$ with the maximum $RC_{\downarrow}[v_i]$ among all the cases. In addition, if the optimal $W_{\downarrow}[v_i]$ is obtained from the assignment of v_i to a child $\hat{u}$, we add a pointer from v_i to $\hat{u}$, and furthermore if $\hat{u}$ was undetermined, we record which assignment, upward or downward, of $\hat{u}$ in Case (2-2-2) yields the optimal $W_{\downarrow}[v_i]$.

To be more precise, we will compute the demand assignment of v_i at each iteration i, $1 \le i \le n$. Suppose v_i is a leaf. We just check the undetermined condition of v_i, and classify its status by Lemma 3.1. Suppose v_i is a non-leaf vertex. We not only check the undetermined condition of v_i by Lemma 3.1, but also *reconsider those undetermined children*, if necessary, of v_i. Suppose v_j is the child of v_i which gave the optimal $W_{\downarrow}[v_i]$, $j < i$. We discuss how to handle those undetermined children of v_i as follows. If v_i is determined to be assigned to its parent or itself, we assign all its undetermined children upward to v_i. If v_i is determined to be assigned downward to v_j, we assign all the undetermined children $u \ne v_j$ to v_i. If v_j itself is undetermined, we then decide the assignment of v_j, upward or downward, depending on which gave the optimal $W_{\downarrow}[v_i]$. On the other hand, if v_i is undetermined, then all the undetermined children u, $u \ne v_j$, of v_i, where v_j gave the optimal $W_{\downarrow}[v_i]$, are assigned upward to v_i similarly. The procedure guarantees that after each iteration i, all the undetermined children of v_i are determined if v_i is determined, and there is at most one undetermined child if v_i is undetermined, a proof of which will be given later. Based on this, we can backtrack by pointers to the undetermined descendants downward and determine their assignment by repeating the same procedure. The following results characterize the invariant condition at each iteration.

Lemma 3.2 *If v_i is classified as undetermined at iteration i, then at most one child of v_i remains undetermined.*

Proof Assume v_j is the child of v_i which gave the optimal $W_{\downarrow}[v_i]$. For every undetermined child u of v_i, i.e., $W_{\uparrow}[u] = W_{\downarrow}[u]$ and $RC_{\uparrow}[u] < RC_{\downarrow}[u]$, $u \ne v_j$, since the residue capacity of u is useless, we can assign u upward to v_i to provide more residue capacity for v_i. Thus there is at most one undetermined child of v_i, i.e., v_j possibly, after iteration i. □

Lemma 3.3 *If v_i is classified as determined at iteration i, then each child of v_i is determined as well.*

Proof It is similar to the proof of Lemma 3.2. If v_i is determined to be assigned downward to one of its children, v_j, we assign to v_i every undetermined child u of v_i, excluding v_j. As for v_j's assignment, since we have computed both cases of $W_{\uparrow}[v_j]$ and $W_{\downarrow}[v_j]$ to obtain the optimal $W_{\downarrow}[v_i]$, v_j is determined as well accordingly. On the other hand, if v_i is determined to be assigned to its parent or to itself, we assign every undetermined child u of v_i upward to v_i because of the same reason as above (the residue capacity of u is useless and the assignment provides more residue capacity of v_i). □

According to the above two lemmas, the next corollary is immediate.

Corollary 3.4 (Invariant Condition) *After each iteration i, $1 \le i \le n$, there is at most one undetermined path consisting of all undetermined vertices in T_{v_i} starting with v_i.*

Based on the above lemmas, given a tree $T = (V, E)$ with a postorder tree traversal $v_1, v_2, \ldots, v_n$, the ALGORITHM MCDUT given in Appendix solves the capacitated domination problem with unsplittable demand on T. We give the following theorem and remark that this algorithm can be modified to output the optimal capacitated dominating multi-set ($x(\delta) = \lceil \frac{\sum_{v \in N[\delta]} f(v,\delta)}{c(\delta)} \rceil$, $\forall \delta \in D$) as well.

Theorem 3.5 ALGORITHM MCDUT *solves the capacitated domination problem with unsplittable demand on trees in linear time.*

Proof The correctness is by induction based on Lemma 3.1 and Corollary 3.4. At each iteration i, $1 \le i \le n$, the assignment of v_i is either determined or undetermined. In the latter case there is at most one path consisting of undetermined vertices starting with v_i in T_{v_i} by Corollary 3.4. Once v_i is determined, we can backtrack by the pointers to obtain the demand assignment for all its undetermined descendants in T_{v_i}. At the last iteration the root v_n is determined since $W_{\uparrow}[v_n] = \infty > W_{\downarrow}[v_n]$ and all the demand assignments are thus obtained. The time cost of computing $W_{\uparrow}[v_i]$, $W_{\downarrow}[v_i]$, $RC_{\uparrow}[v_i]$, $RC_{\downarrow}[v_i]$ and the pointer, is $O(|Ch(v_i)|)$ at each iteration i, where $Ch(v_i)$ is the child set of v_i. If v_i is determined, the demand assignment of v_i takes constant time. Otherwise, if v_i is undetermined, our algorithm backtracks to decide the demand assignment of v_i only once by Lemma 3.3 and Corollary 3.4. The backtracking technique thus takes at most $O(|V|)$ time entirely in amortized analysis, where $|V|$ is the size of T. □

3.2 NP-Completeness for Splittable Demand Model

We next consider the case when the vertex demand can be split. In fact, we show the problem under this model is NP-complete even when the vertex capacity and demand are integers. We shall make a reduction from the decision problem *Subset Sum*, which is a well-known NP-complete problem, to the decision version of capacitated domination problem with splittable demand on trees. The Subset Sum problem is defined as follows.

Given a finite set $S = \{a_1, a_2, \ldots, a_n\}$, $\forall a_i \in \mathbb{Z}^+$, and $W \in \mathbb{Z}^+$, the decision Subset Sum problem is to ask if there exists a subset A of $\{1, 2, \ldots, n\}$ such that $\sum_{i \in A} a_i = W$. Without loss of generality, we assume that $\sum_{i=1}^{n} a_i \ge W$ in this subsection.

Let $M = (\sum_{i=1}^{n} a_i) + 1$, $W' = M \cdot W$, and $a'_i = M \cdot a_i$. Given an instance of the decision Subset Sum problem, we build a capacitated domination problem instance T, where T is a tree consisting of $n + 2$ vertices, where v_{n+1} is the root with capacity 1 and demand W, v_0 is the only child of v_{n+1} with capacity M and demand $\sum_{i=1}^{n} a'_i - W'$, and v_0 has as children, $v_1, v_2, \ldots, v_n$, which are leaf vertices and each leaf v_i has capacity $M + a'_i - a_i$ and demand $M - a_i$.

Lemma 3.6 *Let D be a feasible capacitated dominating multi-set of T, and let $A = \{i \mid v_i \notin D\}$. The number of copies of v_0 in D is no less than $|A|$.*

Proof Let k be the number of copies of v_0. Suppose that $k < |A|$. The capacity at v_0 must satisfy the demand of vertices in A. Thus we have $k \cdot M \geq \sum_{i \in A}(M - a_i)$, which leads to $\sum_{i \in A} a_i - (|A| - k)M \geq 0$, a contradiction since $|A| - k \geq 1$ and $M > \sum_{i=1}^{n} a_i$. □

Corollary 3.7 *The cardinality of a feasible capacitated dominating multi-set of T is no less than n.*

Lemma 3.8 *There exists a subset A of $\{1, 2, \ldots, n\}$ such that $\sum_{i \in A} a_i = W$ if and only if there exists a feasible capacitated dominating multi-set D of size n for T.*

Proof If there is a subset A of $\{1, 2, \ldots, n\}$ satisfying $\sum_{i \in A} a_i = W$, then we construct a capacitated dominating multi-set D as follows. For each $i \in A$, we have a copy of v_0 and assign the demand of v_i to v_0. Each such assignment leaves the residue capacity a_i at v_0. We then assign the demand of v_{n+1} to v_0 since $\sum_{i \in A} a_i = W$.

For $i \notin A$, $i = 1, 2, \ldots, n$, we have a copy of v_i and assign the demand of v_i to itself. Each such assignment leaves residue capacity a'_i at v_i for $i \notin A$. The demand of v_0 is $\sum_{i=1}^{n} a'_i - W' = \sum_{i=1}^{n} a'_i - \sum_{i \in A} a'_i = \sum_{i \notin A} a'_i$, which can be satisfied exactly by the sum of the residue capacity of v_i, $i \notin A$. Thus, we have a feasible capacitated dominating multi-set D of size n.

On the other hand, if there is a feasible capacitated dominating multi-set D of size n for the instance T, define A as $\{i \mid v_i \notin D\}$. By Lemma 3.6 we have at least $|A|$ copies of v_0 and one copy each for the set $\{v_1, \ldots, v_n\} \setminus A$ of vertices. Thus the demand of v_{n+1} is assigned to v_0 in D and we have $\sum_{i \in A} a_i \geq W$. The demand of v_0 is satisfied by the residue capacity of $\{v_0\} \cup \{v_1, \ldots, v_n\} \setminus A$. We have $(\sum_{i \notin A} a'_i) + (\sum_{i \in A} a_i - W) \geq (\sum_{i=1}^{n} a'_i - W')$, which leads to $\sum_{i \in A} a_i \leq W$. Hence we have $\sum_{i \in A} a_i = W$ and completes the proof. □

By Corollary 3.7 and Lemma 3.8, there exists a subset A of $\{1, 2, \ldots, n\}$ such that $\sum_{i \in A} a_i = W$ if and only if the cardinality of the optimal capacitated dominating multi-set D for T is n. In addition, it is easy to show that the capacitated domination problem with splittable demand on trees is in NP. We can verify whether a given solution without demand assignment is feasible in polynomial time in a bottom-up manner. Given a postorder tree traversal, for every vertex we serve its demand by exhausting the sum of the residue capacity of its children first, if any, then its own available capacity, and finally the available capacity of its parent. If it is not enough, the solution is not feasible. Otherwise, we continue to serve the next one. We have the following theorem immediately.

Theorem 3.9 *The capacitated domination problem with splittable demand on trees is NP-complete, even when the vertex capacity and demand are integers.*

3.3 A Pseudo-Polynomial Time Algorithm for Splittable Demand Model

The reduction in the previous subsection points out the difficulty of the capacitated domination problem with splittable demand on trees. Based on the transformation

from *Subset Sum* problem to capacitated domination problem with splittable demand model, it is conceivable [19], but not explicitly, that there exists a pseudo-polynomial time algorithm for its splittable demand model. The main idea is similar to the linear time algorithm provided in Sect. 3.1.

Given a tree $T = (V, E)$ with a postorder tree traversal and $v \in V$, let $Ch(v)$ be the child set of v, $p(v)$ be the parent of v, and T_v denote the subtree rooted at v. We consider a stronger version of the original problem to find a capacitated dominating multi-set D with the maximum residue capacity at the root. The algorithm runs in a bottom-up manner and it processes one vertex at each iteration in the postorder tree traversal.

At each iteration i, we maintain the *residue demand* of v_i, denoted by $RD[v_i]$, and the residue capacity of v_i, denoted by $RC[v_i]$. Initially we have $RD[v_i] = d(v_i)$ and $RC[v_i] = 0$ for every $v_i \in V$. As the vertex v_i is considered, the algorithm first uses up all the available residue capacity, if any, from $Ch(v_i)$ and then from $\{v_i\}$, attempting to assign as much $RD[v_i]$ as possible so that either $RD[v_i]$ is satisfied or all the available residue capacity is exhausted. We shall describe *the first step, exhausting all the available residue capacity* from $Ch(v_i) \cup \{v_i\}$, in detail later.

After the first step, if $RD[v_i]$ is satisfied, then $RC[v_i]$ is also maximized. Otherwise, if $RD[v_i] \neq 0$, that is, the available residue capacity is used up but $RD[v_i]$ is not satisfied, we process $RD[v_i]$ in a greedy manner as follows. Find $u^* \in N[v_i]$ such that $c(u^*)$ is $\max_{u \in N[v_i]}\{c(u)\}$. Create $\lfloor RD[v_i]/c(u^*) \rfloor$ copies of $u^* \in D$ to meet the demand $RD[v_i]$ of v_i. We then have for v_i a new residue demand $RD[v_i]$ which is $RD[v_i]$ mod $c(u^*)$. The algorithm further decides whether the new $RD[v_i]$ can be determined immediately. We recall that Lemma 3.1 characterizes the *undetermined condition* at each iteration i in the unsplittable demand model. A similar *undetermined condition* also holds in the splittable demand model.

Lemma 3.10 *Given a tree $T = (V, E)$ with a postorder tree traversal $v_1, v_2, \ldots, v_n$, we can at each iteration i, $1 \leq i \leq n$, immediately determine the optimal assignment of the residue demand $RD[v_i]$ of v_i, except for the case when $RD[v_i] \leq c(p(v_i)) < c(v_i)$ (this is called the undetermined condition of v_i).*

Proof The argument is similar to Lemma 3.1. After the first step, we have $RD[v_i] \leq c(u^*)$, where $c(u^*) = \max_{v \in N[v_i]}\{c(v)\}$. If $RD[v_i] > \max\{c(p(v_i)), c(v_i)\}$, then $u^* \in Ch(v_i)$ and we immediately assign $RD[v_i]$ to u^*. Otherwise, the demand assignment is easy to do in case $\max\{c(p(v_i)), c(v_i)\} \geq RD[v_i] > \min\{c(p(v_i)), c(v_i)\}$. Finally, if $RD[v_i] \leq c(v_i) \leq c(p(v_i))$, we assign $RD[v_i]$ upward to $p(v_i)$ since it provides more residue capacity at $p(v_i)$. □

If v_i can be determined immediately, then we have $RD[v_i] = 0$. Otherwise we have $RC[v_i] = 0$ by the first step and mark v_i as *undetermined*. The two invariant conditions hold after each iteration i, $1 \leq i \leq n$, (1) $RD[v_i] \cdot RC[v_i] = 0$; and (2) $RD[v_i] \neq 0$ only if $RD[v_i] \leq c(p(v_i)) < c(v_i)$.

Now we describe *the first step*, *exhausting all the available residue capacity* from $Ch(v_i) \cup \{v_i\}$, of our algorithm in detail. When the vertex v_i is considered, the algorithm exhausts the available residue capacity from $Ch(v_i)$ first. Let

$Ch_d(v_i) = \{p_1, p_2, \ldots, p_l\}$ be the set of *determined* children of v_i and $Ch_u(v_i) = \{q_1, q_2, \ldots, q_m\}$ be the set of *undetermined* children of v_i. Note that $RD[p_j] = 0$, $1 \le j \le l$, and $RD[q_k] \le c(v_i) < c(q_k)$, $1 \le k \le m$. There may be some residue capacity at v_i contributed by those determined children which are assigned upward. The first step has the following two phases.

Phase (1) We exhaust the total residue capacity in $Ch_d(v_i)$, i.e., $\sum_{p_j \in Ch_d(v_i)} RC[p_j]$. It is obvious that the residue capacity $RC[p_j]$ is useless after the iteration i, $\forall p_j \in Ch_d(v_i)$. We assign $RD[v_i]$ to $p_1, p_2, \ldots, p_l$ as far as possible. If $RD[v_i] \le \sum_{p_j \in Ch_d(v_i)} RC[p_j]$, then $RD[v_i]$ can be satisfied. We then assign $q_1, q_2, \ldots, q_m$ upward to v_i since it provides $p(v_i)$ more residue capacity. The first step is done. Otherwise, if $RD[v_i] > \sum_{p_j \in Ch_d(v_i)} RC[p_j]$, we update $RD[v_i] = RD[v_i] - \sum_{p_j \in Ch_d(v_i)} RC[p_j]$ and enter the next phase.

Phase (2) We arrange the demand assignment of the vertices in $Ch_u(v_i)$ to satisfy $RD[v_i]$ while $RC[v_i]$ is maximized.
If $RD[v_i] > \sum_{q_k \in Ch_u(v_i)} (c(q_k) - RD[q_k])$, then it is impossible that $RD[v_i]$ can be satisfied by any arrangement of the total available residue capacity of $Ch_u(v_i)$, since by the undetermined condition of q_k, $c(q_k) > c(v_i)$ for each $q_k \in Ch_u(v_i)$ and we assign $q_1, q_2, \ldots, q_m$ downward to themselves. Subsequently, we use up all available residue capacity in $Ch_u(v_i)$ to satisfy $RD[v_i]$, and then assign the rest of $RD[v_i]$, $RD[v_i] - \sum_{q_k \in Ch_u(v_i)} (c(q_k) - RD[q_k])$, to itself with the residue capacity of v_i, $RC[v_i]$. This complete the *first step*.

Otherwise, i.e., $RD[v_i] \le \sum_{q_k \in Ch_u(v_i)} (c(q_k) - RD[q_k])$, we will proceed to arrange the demand assignment of the vertices in $Ch_u(v_i)$ to satisfy $RD[v_i]$ while maximizing $RC[v_i]$. This problem reduces to the following RELAXED KNAPSACK PROBLEM, which is a variation of the well-known *Knapsack Problem*. Note that the first invariant condition, that is, $RD[v_i]$ is satisfied or $RC[v_i]$ is exhausted, holds in both phases.

Definition 3.11 (The Relaxed Knapsack Problem) Given a set of m ordered pairs (a_k, b_k) denoting respectively the size and profit of the kth item, $\forall a_k, b_k \in \mathbb{Z}^+ \cup \{0\}$, $1 \le k \le m$, and a nonnegative integer W, find a subset $A \subseteq \{1, 2, \ldots, m\}$ such that $\sum_{k \in A} b_k - \max\{0, \sum_{k \in A} a_k - W\}$ is maximized.

We select the items and place them into a knapsack of the relaxed size W to maximize the sum of the profit $\sum_{k \in A} b_k$. In particular, the additional compensation $\sum_{k \in A} a_k - W$ is required if the total size of the selected items, $\sum_{k \in A} a_k$, exceeds W. We shall present an $O(m^2 M)$ pseudo-polynomial time algorithm, where $M = \max_{1 \le k \le m}\{b_k\}$, to solve the RELAXED KNAPSACK PROBLEM at the end of this subsection.

The transformation is done in linear time as follows. Let $W = \sum_{q_k \in Ch_u(v_i)} (c(q_k) - RD[q_k]) - RD[v_i]$, $a_k = c(q_k) - RD[q_k]$, and $b_k = c(v_i) - RD[q_k]$, for $1 \le k \le m$. We illustrate this further in detail. We know each vertex q_k in $Ch_u(v_i)$ has $c(q_k) > c(v_i)$. Thus, it provides more residue capacity for $RD[v_i]$ to assign $RD[q_k]$ to q_k itself than to assign $RD[q_k]$ to v_i. If we assign every vertex in $Ch_u(v_i)$ downward to itself, then

the sum of the residue capacity in $Ch_u(v_i)$ is $\sum_{q_k \in Ch_u(v_i)}(c(q_k) - RD[q_k]) - RD[v_i]$ after $RD[v_i]$ is satisfied. This is the relaxed size W of the knapsack. If we assign $RD[q_k]$, $q_k \in Ch_u(v_i)$, upward to v_i, the knapsack size is decreased by $a_k = c(q_k) - RD[q_k]$ and the total profit is increased by $b_k = c(v_i) - RD[q_k]$. Therefore, there is a subset $A \subseteq \{1, 2, \ldots, m\}$ such that $\sum_{k \in A} b_k - \max\{0, \sum_{k \in A} a_k - W\}$ is maximized if and only if there exists an optimal demand assignment of the vertices in $Ch_u(v_i)$ such that $RD[v_i]$ can be satisfied while maximizing $RC[v_i]$ for Phase (2) of the first step.

Theorem 3.12 *The demand assignment of vertices in $Ch_u(v_i)$ can be optimally determined in $O(|Ch_u(v_i)|^2 c(v_i))$ time such that $RD[v_i]$ is satisfied while maximizing $RC[v_i]$, for Phase (2) of the first step at each iteration i, $1 \le i \le n$.*

According to Lemma 3.10 and Theorem 3.12, the following corollary is immediate.

Corollary 3.13 *There exists at most one undetermined vertex, namely v_i, in T_{v_i}, after each iteration i, $1 \le i \le n$.*

Based on Theorem 3.12 and Corollary 3.13, the time needed at each iteration i is $O(|Ch(v_i)|^2 c(v_i)) + O(|Ch(v_i)|) = O(|Ch(v_i)|^2 c(v_i))$. The overall time complexity is $\sum_{v_i \in V} O(c(v_i)|Ch(v_i)|^2) = O(C|V|^2)$, where $C = \max_{v_i \in V} c(v_i)$ is the maximum capacity. The ALGORITHM MCDST is presented in Appendix.

Theorem 3.14 *Given a tree $T = (V, E)$ with a postorder tree traversal,* ALGORITHM MCDST *solves the capacitated domination problem with splittable demand on T in $O(C|V|^2)$ time, where C is the maximum capacity.*

3.3.1 The Relaxed Knapsack Problem

We describe below the algorithm DP_RKP based on dynamic programming to solve the RELAXED KNAPSACK PROBLEM.

Let $MinW(k, B)$ denote the minimum total size among all the combinations of the first k items achieving the total profit B exactly. We let $MinW(k, B) = \infty$ if no such combination exists. Let $M = \max_{1 \le k \le m}\{b_k\}$, and $MinW(0, B) = \infty$ initially for $0 \le B \le mM$.

For each (k, B) with $1 \le k \le m$ and $0 \le B \le mM$, the algorithm computes $MinW(k, B)$ based on $MinW(k-1, B')$, $0 \le B' \le mM$, by considering the following three cases: (1) the kth item is not picked by the algorithm; (2) the kth item is picked and the total size we picked does not exceed W; (3) the kth item is picked and the total size we picked exceeds W. The recurrence formula for $MinW(k, B)$ is

summarized as follows.

$$MinW(k, B) = \min \begin{cases} MinW(k-1, B), \\ MinW(k-1, B-b_k) + a_k, \\ \quad \text{if } B \geq b_k \text{ and } MinW(k-1, B-b_k) + a_k \leq W, \\ \min\{MinW(k-1, B') + a_k \mid MinW(k-1, B') + a_k > W \text{ and} \\ \quad B = (B' + b_k) - (MinW(k-1, B') + a_k - W), \\ \quad \text{for each } 0 \leq B' \leq mM\}. \end{cases}$$

At each iteration k, we compute $MinW(k, B)$ for every $0 \leq B \leq mM$. As $MinW(m, B)$ is obtained for each $0 \leq B \leq mM$, the algorithm outputs the maximum total profit B^* with $MinW(m, B^*) < \infty$.

In addition, by using a backtracking technique, we can modify the algorithm to output the solution $A \subseteq \{1, 2, \ldots, m\}$ as well.

Lemma 3.15 *The* RELAXED KNAPSACK PROBLEM *can be solved by dynamic programming in* $O(m^2M)$ *time.*

Proof The correctness is obvious based on the recurrence formula. For each $MinW(k, B)$, the straightforward approach to computing the third case of the formula takes $O(mM)$ time.

However, we can reduce the time cost by pre-considering the third case of $MinW(k+1, B_0)$ after $MinW(k, B)$ is computed, where $B_0 = (B + b_{k+1}) - (MinW(k, B) + a_{k+1} - W)$. To be precise, when $MinW(k, B)$ is computed, we check that whether $MinW(k, B) + a_{k+1}$ exceeds W. If it does, we update $MinW(k+1, B_0)$ in advance. Otherwise, we do nothing. This reduces the time cost of the third case to amortized $O(1)$ time. The overall time needed is therefore $O(m \cdot mM) = O(m^2M)$. □

4 Approximation Algorithms

We investigate approximation algorithms for capacitated domination problem with demand constraints. In Sect. 4.1, we recall the splittable demand model on trees. We first review the RELAXED KNAPSACK PROBLEM and develop a fully polynomial time approximation scheme (FPTAS) for this problem. Based on the FPTAS result, we give a polynomial time approximation scheme (PTAS) for the capacitated domination problem with splittable demand on trees. In Sect. 4.2, we consider weighted capacitated domination problem with splittable demand on general graphs and present a primal-dual algorithm [9, 16, 20] that gives a Δ^*-approximation, where Δ^* is the closed degree of the input graph.

4.1 Approximation on Trees

We showed that the capacitated domination problem with splittable demand on trees is NP-complete in Sect. 3.2, even when the vertex capacity and demand are integers.

By our pseudo-polynomial time algorithm for the RELAXED KNAPSACK PROBLEM, we further showed that it can be solved in pseudo-polynomial time for the integer version in Sect. 3.3. In this subsection, we first give a fully polynomial time approximation scheme (FPTAS) for the RELAXED KNAPSACK PROBLEM. The idea is similar to the well-known FPTAS result of Ibarra and Kim [19] for the *Knapsack Problem*. Based on our FPTAS result, we then present a polynomial time approximation scheme for the capacitated domination problem with splittable demand on trees.

4.1.1 FPTAS for the Relaxed Knapsack Problem

Given a Relaxed Knapsack Problem instance, (a_i, b_i), $1 \le i \le n$, where a_i and b_i are the size and the profit of the ith item, and $W \ge 0$, let $M = \max_{1 \le i \le n}\{b_i\}$ be the maximum profit. We re-scale the input to apply the dynamic programming algorithm DP_RKP provided in Sect. 3.3.1.

Algorithm FPTAS_RKP (FPTAS for RELAXED KNAPSACK PROBLEM)

1. Given $\epsilon > 0$, let $k = \frac{\epsilon M}{2n+1}$, $a'_i = \lceil \frac{a_i}{k} \rceil$, $b'_i = \lfloor \frac{b_i}{k} \rfloor$, and $W' = \lfloor \frac{W}{k} \rfloor$.
2. Use (a'_i, b'_i), $1 \le i \le n$, and W' as input parameters of the new Relaxed Knapsack Problem instance.
   Apply DP_RKP given in Sect. 3.3.1 to obtain a maximum total profit B^*.
3. Output the solution A^* by the backtracking technique from B^*.

Note that, without loss of generality, we may assume $k \ge 1$ since if it is not the case, we simply apply DP_RKP without re-scaling. The solution we obtained for the given Relaxed Knapsack Problem instance will be optimal and the time required will be no more than that of FPTAS_RKP required.

We have the following proposition. The proof is straightforward and hence omitted.

Proposition 4.1 *We have* $r \le s\lceil \frac{r}{s} \rceil \le r + s$ *and* $r - s \le s\lfloor \frac{r}{s} \rfloor \le r$, *for every* $r, s \in R$ *with* $s > 0$.

For any specified solution A, we denote the total profit of the original problem instance by $\text{profit}(A) = \sum_{i \in A} b_i - \max\{0, \sum_{i \in A} a_i - W\}$ and the total profit of the new problem instance by $\text{profit}'(A) = \sum_{i \in A} b'_i - \max\{0, \sum_{i \in A} a'_i - W'\}$. We have the following lemma.

Lemma 4.2 *Let* A^* *be the solution obtained by Algorithm* FPTAS_RKP. *We have* $\text{profit}(A^*) \ge (1 - \epsilon) \cdot \text{profit}(O)$, *where* O *is the optimal solution of the original problem.*

Proof Let A be any feasible solution. We have $0 \le \text{profit}(A) - k \cdot \text{profit}'(A) = \sum_{i \in A}(b_i - k\lfloor \frac{b_i}{k} \rfloor) + (\max\{0, \sum_{i \in A} k\lceil \frac{a_i}{k} \rceil - k\lfloor \frac{W}{k} \rfloor\} - \max\{0, \sum_{i \in A} a_i - W\})$. By Proposition 4.1, we can obtain $\sum_{i \in A}(b_i - k\lfloor \frac{b_i}{k} \rfloor) \le nk$ and $0 \le (\sum_{i \in A} k\lceil \frac{a_i}{k} \rceil - k\lfloor \frac{W}{k} \rfloor) - (\sum_{i \in A} a_i - W) \le (n + 1)k$.

We then consider the item $\max\{0, \sum_{i\in A} k\lceil\frac{a_i}{k}\rceil - k\lfloor\frac{W}{k}\rfloor\} - \max\{0, \sum_{i\in A} a_i - W\}$ in the following. Since $\sum_{i\in A} k\lceil\frac{a_i}{k}\rceil - k\lfloor\frac{W}{k}\rfloor \geq \sum_{i\in A} a_i - W$, the inequality $\sum_{i\in A} k\lceil\frac{a_i}{k}\rceil - k\lfloor\frac{W}{k}\rfloor \leq 0$ would imply $\sum_{i\in A} a_i - W \leq 0$. Furthermore, the inequality $\sum_{i\in A} a_i - W \leq 0$ would imply $(\sum_{i\in A} k\lceil\frac{a_i}{k}\rceil - k\lfloor\frac{W}{k}\rfloor) - 0 \leq (\sum_{i\in A} k\lceil\frac{a_i}{k}\rceil - k\lfloor\frac{W}{k}\rfloor) - (\sum_{i\in A} a_i - W) \leq (n+1)k$. Therefore we can obtain $0 \leq \text{profit}(A) - k\cdot \text{profit}'(A) \leq nk + (n+1)k = k(2n+1)$.

On the other hand, since A^* is the optimal solution of the new problem instance, we have $\text{profit}(A^*) \geq k\cdot \text{profit}'(A^*) \geq k\cdot \text{profit}'(O) \geq \text{profit}(O) - k(2n+1) = \text{profit}(O) - \epsilon M \geq (1-\epsilon)\cdot \text{profit}(O)$, which proves the lemma. □

By Lemma 4.2, the profit of the solution obtained is greater than or equal to $(1-\epsilon)\cdot$ profit(O), where O is the optimal solution. The running time of the algorithm is $O(n^2\lfloor\frac{M}{k}\rfloor) = O(n^2\lfloor\frac{n}{\epsilon}\rfloor)$ by Lemma 3.15. The following theorem is immediate.

Theorem 4.3 *Algorithm* FPTAS_RKP *is an FPTAS for the* Relaxed Knapsack Problem *which gives a* $(1-\epsilon)$*-approximation in* $O(n^2\lfloor\frac{n}{\epsilon}\rfloor)$ *time.*

4.1.2 A Simple 2-Approximation on Trees

Based on our FPTAS result of the Relaxed Knapsack Problem, we provide a simple 2-approximation algorithm for the capacitated domination problem with splittable demand on a tree $T = (V, E)$ in $O(|V|^3\Delta)$ time, where Δ is the maximum degree, as follows. Given a capacitated domination problem instance, let $\epsilon = \frac{1}{\Delta}$. Consider Phase (2) of our first step for every $v_i \in V$ in Sect. 3.3. We apply Algorithm FPTAS_RKP instead of DP_RKP to arrange the demand assignment of vertices in $Ch_u(v_i)$, and have an additional copy of v_i after the arrangement. Note that we have this copy only when FPTAS_RKP is applied.

Lemma 4.4 *The approximation algorithm based on* FPTAS_RKP *yields a* 2*-approximation factor for the capacitated domination problem with splittable demand on a tree* $T = (V, E)$ *in* $O(|V|^3\Delta)$ *time, where* Δ *is the maximum degree.*

Proof For any vertex $v_i \in V$, the maximum residue capacity of v_i is at most $|Ch_u(v_i)|\cdot c(v_i)$ after arranging the demand assignment of $Ch_u(v_i)$. By the choice of ϵ, the error produced by FPTAS_RKP is at most $\epsilon\cdot|Ch_u(v_i)|\cdot c(v_i) \leq c(v_i)$, which can be supplemented by an additional copy. The approximation ratio is therefore at most 2 for each vertex. The overall time complexity is $\sum_{v_i\in V}(O(|Ch(v_i)|) + O(|Ch_u(v_i)|^3\frac{1}{\epsilon})) = O(|V|^3\Delta)$. □

4.1.3 PTAS on Trees

The basic idea of the 2-approximation algorithm extends to a polynomial time approximation scheme for the capacitated domination with splittable demand on trees. Given an integer constant $k > 0$, the approach is to apply FPTAS_RKP only when it requires at least k copies to satisfy the arrangement of the demand assignment of $Ch_u(v_i)$ in Phase (2) of our first step in Sect. 3.3.

The process is done as follows. For $j = 1, 2, \ldots, k$, we verify that whether j copies are sufficient for the demand assignment of $Ch_u(v_i)$. If it holds, then we have found the optimal arrangement, which corresponds to the smallest such j, and no approximation is required at this stage. Otherwise we apply FPTAS_RKP to arrange $Ch_u(v_i)$ and have an additional copy of v_i. The approximation ratio is therefore $\frac{k+1}{k}$. For the time complexity, it takes $(O\sum_{j=1}^{k}\binom{|Ch_u(v_i)|}{j}) = O(|Ch_u(v_i)|^k)$ time to verify the number of copies required in the arrangement of $Ch_u(v_i)$. The overall time is $\sum_{v_i \in V}(O(|Ch(v_i)|) + |Ch_u(v_i)|^k + O(|Ch_u(v_i)|^3 \frac{1}{\epsilon})) = O(|V|^k + |V|^3\Delta)$.

Theorem 4.5 *We have a polynomial time approximation scheme for the capacitated domination problem with splittable demand on trees that computes a* $\frac{k+1}{k}$*-approximation solution in* $O(|V|^k + |V|^3\Delta)$ *time, where* k *is an integer constant and* Δ *is the maximum degree.*

4.2 General Graphs

In this subsection, we present a primal-dual algorithm [9, 16, 20] that gives a Δ^*-approximation for weighted capacitated domination with splittable demand on general graphs, where Δ^* is the closed degree of the input graph. The algorithm is based on the dual fitting technique.

An integer linear programming (ILP) of this problem is written in (1). A feasible primal solution corresponds to a feasible capacitated dominating multi-set on a given weighted graph. The additional constraint $d(v_j)x(v_i) - f(v_j, v_i) \geq 0$, which is unnecessary in the ILP formulation, is required to bound the integrality gap between the fractional optimum and the integral optimum in the relaxation. Actually the integrality gap could be large without this constraint. For example, consider a star which consists of n vertices with uniform capacity n, uniform demand 1, and uniform weight 1. The integral optimum is simply to have one copy at the center. However, there could be one optimal fractional solution with $1/n$ copies placed at each vertex, giving one copy at each vertex after rounding.

$$
\begin{aligned}
\text{Minimize} \quad & \sum_{i=1}^{n} w(v_i)x(v_i) \\
& c(v_i)x(v_i) - \sum_{v_j \in N[v_i]} f(v_j, v_i) \geq 0, \quad i = 1, 2, \ldots, n \\
& \sum_{v_j \in N[v_i]} f(v_i, v_j) \geq d(v_i), \quad i = 1, 2, \ldots, n \\
& d(v_j)x(v_i) - f(v_j, v_i) \geq 0, \quad v_j \in N[v_i],\ i = 1, 2, \ldots, n \\
& x(v_i) \in \mathbb{Z}^+ \cup \{0\}, \quad i = 1, 2, \ldots, n
\end{aligned}
\tag{1}
$$

The dual program of the relaxation of (1) is described in (2). The objective value of a dual feasible solution is a lower bound of any integral optimum of the primal program. Based on the dual fitting technique, we increase every y_i simultaneously as large as possible to maximize the objective function, while we maintain the dual

feasibility.

$$\begin{aligned}
\text{Maximize} \quad & \sum_{i=1}^{n} d(v_i) y_i \\
& c(v_i) z_i + \sum_{v_j \in N[v_i]} d(v_j) g_{i,j} \le w(v_i), \quad i = 1, 2, \ldots, n \\
& y_i \le z_j + g_{j,i}, \qquad v_j \in N[v_i], \quad i = 1, 2, \ldots, n \\
& y_i \ge 0, \qquad z_i \ge 0, \qquad g_{j,i} \ge 0, \qquad v_j \in N[v_i], \quad i = 1, 2, \ldots, n
\end{aligned} \tag{2}$$

Given a weighted graph $G = (V, E)$, let $V^\phi \subseteq V$ denote the set of vertices whose demands have not been assigned yet. For every vertex v, we use $d_N^\phi(v)$ to denote the total amount of unassigned demand of all the closed neighbors of v. In addition, we define the inequality $d_N^\phi(v) \le c(v)$ to be the *critical condition* of v and $D^c_{u\bar{v}}$ to be the amount of demand of a closed neighbor u of v, $d(u)$, which is to be assigned to vertices other than v after the critical condition of v holds. Now we describe our ALGORITHM MCDAWG (see Appendix) briefly. Initially no demand is assigned and we mark all the vertices *closed*. As the algorithm runs, several vertices are marked *open* while their weight constraints of the dual program are met with equality. As soon as one vertex v is marked *open*, all unassigned demands of the closed neighbors of v are assigned to v (the amount is $d_N^\phi(v)$). Subsequently, some demand of the closed neighbor, say u, of v ($u = v$ possibly), which was previously assigned to v, gets reassigned to a closed neighbor w of u, $w \neq v$, when w is marked *open* and the critical condition of w holds (the reassigned amount is the $f(u, v)$ portion of $D^c_{u\bar{w}}$). All the demand assignment can be arranged accordingly, and finally the capacitated dominating multi-set is $D = \{v;\ v \text{ is marked open}\}$ and $x(v) = \lceil (\sum_{u \in N[v]} f(u, v))/c(v) \rceil$, $\forall v \in D$.

The main process is described in detail as follows. In the dual program, all the dual variables y_i are zero initially. This is a dual feasible solution with all $z_j = 0$ and $g_{j,i} = 0$. We increase all the dual variables y_i simultaneously, for each vertex $v_i \in V^\phi$ whose demand is unassigned. To maintain the dual feasibility of the constraint $y_i \le z_j + g_{j,i}$, we have to increase z_j or $g_{j,i}$, for every $v_j \in N[v_i]$, while we increase y_i. To be more precise, if the closed neighbors of v_j have a large amount of unassigned demands, specifically, $d_N^\phi(v_j) > c(v_j)$, then we increase z_j. Otherwise, we increase $g_{j,i}$. When we increase the dual variables simultaneously for each vertex in V^ϕ, we stop increasing the dual variable v_k as soon as the weight constraint of v_k, $c(v_k) z_k + \sum_{v_\ell \in N[v_k]} d(v_\ell) g_{k,\ell} \le w(v_k)$, is met with equality. We then mark this vertex v_k *open* and assign to v_k $d_N^\phi(v_k)$ unassigned demand from the closed neighbors of v_k. In addition, if the critical condition of v_k, i.e., $d_N^\phi(v_k) \le c(v_k)$, holds at the same time, we also reassign to v_k $D^c_{u\bar{v}_k}$ demand from every closed neighbor u of v_k. Note that for every closed neighbor u of v_k, we have to update $D^c_{u\bar{w}}$, $\forall w \in N[u]$.

Lemma 4.6 *The total weight of each open vertex,* $\sum_{v \in D} w(v) \cdot x(v)$, *can be distributed so that each unit of the demand (say from* v_j*) costs at most* $deg[v_j] \cdot y_j$ *weight, where* $deg[v_j]$ *is the closed degree of* v_j.

Proof We divide the *open* vertices into two groups by the *critical condition*. A vertex v_i is defined to be *light* if the critical condition of v_i, $d_N^\phi(v_i) \le c(v_i)$, holds, when it is marked open, and defined to be *heavy* otherwise. The charging scheme is specified in the following.

Consider a *heavy* vertex v_i. When v_i is marked open, we have $d_N^\phi(v_i) > c(v_i)$ and $c(v_i)z_i + \sum_{v_j \in N[v_i]} d(v_j)g_{i,j} = w(v_i)$, and assign to v_i all $d_N^\phi(v_i)$ unassigned demand of its closed neighbors. In particular, some of these demands, denoted by a subset $R(v_i)$, will be reassigned to other vertices subsequently. Based on our dual fitting rule, we have $g_{i,j} = 0$ and $y_j = z_i$ for $v_j \in N[v_i]$, which implies $w(v_i) = c(v_i)z_i = c(v_i)y_j$. If we want to distribute $w(v_i)$ so that each unit of the demand (say from v_j) costs at most y_j weight, then v_j needs $c(v_i)$ units of demand to be charged, for each copy of v_i.

Suppose $d_N^\phi(v_i) = p \cdot c(v_i) + q$, where $0 \le q < c(v_i)$. If $R(v_i) \ge q$, then we need at most p copies of v_i, and the total weight can be distributed to any $p \cdot c(v_i)$ units of the $d_N^\phi(v_i)$ demand assigned to v_i. Hence, each unit of the $d_N^\phi(v_i)$ demand (say from v_j) assigned to v_i is charged at most y_j weight. If $0 \le R(v_i) < q$, then we need $p+1$ copies of v_i and the total weight needs $(p+1)c(v_i)$ demand to be charged. We first charge all the units of the $d_N^\phi(v_i)$ demand once. We then need to charge $c(v_i) - q$ units of demand. Since there are at least $p \cdot c(v_i)$ un-reassigned units of the $d_N^\phi(v_i)$ demand and $p \ge 1$ (since v_i is heavy and $d_N^\phi(v_i) > c(v_i)$), we charge $c(v_i) - q$ units of the un-reassigned demand again.

On the other hand, if v_i is a *light* vertex, we need only one copy of v_i ($d_N^\phi(v_i) \le c(v_i)$) when v_i is marked open. If v_i is *light* in the beginning, that is, $\sum_{v_j \in N[v_i]} d(v_j) \le c(v_i)$, then we have $z_i = 0$ and $y_j = g_{i,j}$ for $v_j \in N[v_i]$, based on our dual fitting rule. Thus $w(v_i) = \sum_{v_j \in N[v_i]} d(v_j)g_{i,j} = \sum_{v_j \in N[v_i]} d(v_j)y_j$. The weight of only one copy, $w(v_i)$, can be distributed to all the units of $d_N^\phi(v_i)(= \sum_{v_j \in N[v_i]} d(v_j))$ demand assigned to v_i, and each unit of demand (say from v_j) gets a charge of y_j.

Otherwise, if $\sum_{v_j \in N[v_i]} d(v_j) > c(v_i)$ initially, then $d_N^\phi(v_i) = c(v_i)$ at some point in time, as we increase the dual variables simultaneously for each vertex in V^ϕ. After the time, we fix the value of z_i and subsequently increase $g_{i,j}$ for $v_j \in N[v_i] \cap V^\phi$, based on our dual fitting rule. Note that $c(v_i) = d_N^\phi(v_i) + \sum_{v_j \in N[v_i]} D^c_{v_j \bar{v}_i} = \sum_{v_j \in N[v_i] \cap V^\phi} d(v_j) + \sum_{v_j \in N[v_i]} D^c_{v_j \bar{v}_i}$, since we assign the $D^c_{v_j \bar{v}_i}$ portion of $d_N^\phi(v_i)$ demand to vertices other than v_i for $v_j \in N[v_i]$ after the time. Thus, as v_i is marked open, we have $w(v_i) = c(v_i)z_i + \sum_{v_j \in N[v_i]} d(v_j)g_{i,j} = (\sum_{v_j \in N[v_i] \cap V^\phi} d(v_j) + \sum_{v_j \in N[v_i]} D^c_{v_j \bar{v}_i})z_i + (\sum_{v_j \in N[v_i] \cap V^\phi} d(v_j)g_{i,j} + \sum_{v_j \in N[v_i] \setminus V^\phi} d(v_j)g_{i,j}) = \sum_{v_j \in N[v_i] \cap V^\phi} d(v_j)(z_i + g_{i,j}) + (\sum_{v_j \in N[v_i]} D^c_{v_j \bar{v}_i})z_i + \sum_{v_j \in N[v_i] \setminus V^\phi} d(v_j)g_{i,j}$. The first item is equal to $\sum_{v_j \in N[v_i] \cap V^\phi} d(v_j)y_j$ because $y_j = z_i + g_{i,j}$. The second item $\sum_{v_j \in N[v_i]} D^c_{v_j \bar{v}_i} z_i = \sum_{v_j \in N[v_i] \setminus V^\phi} D^c_{v_j \bar{v}_i} z_i$ ($D^c_{v_j \bar{v}_i} = 0$ if $v_j \in V^\phi$) is no larger than $\sum_{v_j \in N[v_i] \setminus V^\phi} d(v_j)z_i$ since $D^c_{v_j \bar{v}_i} \le d(v_j)$, $\forall v_j$. Therefore the weight $w(v_i)$ is no large than $\sum_{v_j \in N[v_i] \cap V^\phi} d(v_j)y_j + \sum_{v_j \in N[v_i] \setminus V^\phi} d(v_j)(z_i + g_{i,j}) =$

$\sum_{v_j \in N[v_i]} d(v_j) y_j$, because $y_j = z_i + g_{i,j}$. Hence each unit of demand (say from v_j) assigned to v_i gets a charge of y_j again.

Finally, for every unit d of demand $d(v_j)$ of a vertex $v_j \in V$, without loss of generality, assume $deg[v_j] > 1$ and let $A \subseteq N[v_j]$ be the set of vertices that have charged d. If $|A| = 1$, then d is charged at most two ($\leq deg[v_j]$) times (by a heavy closed neighbor). If $|A| \geq 2$, then every light vertex in A charges d once, and each heavy vertex in A also charges d once since the demand d is reassigned ($|A| \geq 2$). Thus d gets a charge of $|A| \cdot y_j \leq deg[v_j] \cdot y_j$. □

Theorem 4.7 ALGORITHM MCDAWG *obtains a Δ^*-approximation solution for weighted capacitated domination problem with splittable demand on a general graph $G = (V, E)$, where Δ^* is the maximum closed degree of G.*

Proof Let D^* be the multi-set of *open* vertices returned by ALGORITHM MCDAWG and $w(D^*) = \sum_{v \in D^*} w(v)x(v)$. Since the objective value $\sum_v d(v)y_v$ of a dual feasible solution is a lower bound of any integral optimum of the primal program, by Lemma 4.6, we have $w(D^*) \leq \sum_{v \in V} d(v) \cdot (deg[v]y_v) \leq \Delta^* \cdot \sum_{v \in V} d(v)y_v \leq \Delta^* \cdot w(OPT)$, where OPT is an optimal solution. □

5 Conclusion

In this paper we have presented a linear time algorithm for capacitated domination problem with unsplittable demand on trees. As for the splittable demand model on trees, we have shown that it is NP-complete, and solvable in pseudo-polynomial time. We have also provided two approximation algorithms. One is polynomial time approximation scheme (PTAS) for trees and the other is Δ^*-factor by using primal-dual method for general weighted graphs, where Δ^* is the closed degree of the input graph. Finally we conclude with some comments followed by future work.

5.1 Discussion

Consider the *hard* capacity model. Recall that the *hard* capacity model restricts the number of available copies of each vertex to a given function of vertices. We refer to the result of Chuzhoy and Naor [9], and find that the hard capacitated domination problem with unsplittable demand on graphs is also unapproximable, even on bipartite graphs.

For trees, it is not difficult to show that the hard capacitated domination problem with unsplittable demand is NP-hard. We provide here a reduction from *Subset Sum*. Given a *Subset Sum* instance with parameters W and a_i, $1 \leq i \leq n$, we construct a star T of $n + 1$ vertices. Let $M = \max\{a_i\} + W + 1$. The number of the bounded available copies, capacity, and demand of the center are W, M, and 0, respectively. The capacity and demand of the n leaves are a_i and Ma_i, respectively.

Lemma 5.1 *There is a subset $A \subseteq \{1, \ldots, n\}$ with $\sum_{i \in A} a_i = W$ if and only if the cardinality of the optimal capacitated dominating multi-set on T is $kM + W$, for some k.*

Proof It is trivial for the necessary condition because $M > W$ by the definition of M. As for the sufficient condition, given the cardinality S of any feasible capacitated dominating multi-set, the number of copies in the center is well defined by the remainder of $\frac{S}{M}$, since $M > W$. □

We conjecture that the hard capacitated domination problem with splittable demand on trees is unapproximable. On the other hand, for the soft uniform capacity model, e.g., when each vertex has the same capacity, we believe that the capacitated domination problem with splittable demand on trees can be solved in linear time because no undetermined condition exists for each vertex. However, our linear time algorithm does not extend to the uniform splittable demand model in which each vertex has the same demand.

5.2 Future Work

We present a Δ^*-approximation for the capacitated domination problem with splittable demand on general graphs, which corresponds to the Δ-factor result for the classical domination problem, where Δ is the maximum (open) degree of the graph. However, the approximation for the unsplittable demand model on general graphs still remains open. In contrast to the other $O(\ln n)$-factor approximation result for the classical domination problem, it would not be surprised if there is a similar result for the capacitated domination problem.

Recall the facility location problem. It is natural to introduce the concept of *service cost* to the capacitated domination problem. In addition, it is also interesting to explore the complexity of capacitated domination problem with demand constraints on other graph classes.

Appendix

Algorithm Min_Capacitated_Domination_Unsplittable_demand_on_Trees MCDUT

Input

1. A tree $T = (V, E)$ with a postorder tree traversal $v_1, v_2, \ldots, v_n$
2. A capacity function $c : V \to \mathbb{R}^+ \cup \{0\}$
3. A demand function $d : V \to \mathbb{R}^+ \cup \{0\}$

Output

The minimum cardinality of a capacitated dominating set D of T, $|D| = W_{\downarrow}[v_n]$, and the optimal demand assignment function $f : V \times D \to \mathbb{R}^+ \cup \{0\}$, where $f(v, \hat{u})$ corresponds to each $\hat{u} \in D, \forall v \in V$.

Begin

for $i = 1$ to n

Let $Ch(v_i)$ be the child set of v_i and $p(v_i)$ the parent of v_i.

/* **Case 1:** assume $d(v_i)$ is assigned upward to its parent of v_i, $p(v_i)$. */

if $i < n$ /* v_i is not root. */

$$RC_{\uparrow}[v_i] := \sum_{u \in Ch(v_i),\ W_{\downarrow}[u] \geq W_{\uparrow}[u]} (c(v_i) - (d(u) \bmod c(v_i)))$$

$$W_{\uparrow}[v_i] := \lceil d(v_i)/c(p(v_i)) \rceil + \sum_{u \in Ch(v_i), W_{\downarrow}[u] < W_{\uparrow}[u]} W_{\downarrow}[u] + \sum_{u \in Ch(v_i), W_{\downarrow}[u] \geq W_{\uparrow}[u]} W_{\uparrow}[u] - \lfloor RC_{\uparrow}[v_i]/c(v_i) \rfloor$$

$RC_{\uparrow}[v_i] := RC_{\uparrow}[v_i] \bmod c(v_i)$
end if

/* **Case 2:** assume $d(v_i)$ is assigned to one of $Ch(v_i) \cup \{v_i\}$. */
/* **Case 2-1:** assume $d(v_i)$ is assigned to $\{v_i\}$ itself. */
$RC_{v_i}[v_i] := \sum_{u \in Ch(v_i), W_{\downarrow}[u] \geq W_{\uparrow}[u]} (c(v_i) - (d(u) \bmod c(v_i))) + (c(v_i) - (d(v_i) \bmod c(v_i)))$
$W_{v_i}[v_i] := \lceil d(v_i)/c(v_i) \rceil + \sum_{u \in Ch(v_i), W_{\downarrow}[u] < W_{\uparrow}[u]} W_{\downarrow}[u] + \sum_{u \in Ch(v_i), W_{\downarrow}[u] \geq W_{\uparrow}[u]} W_{\uparrow}[u] - \lfloor RC_{v_i}[v_i]/c(v_i) \rfloor$
$RC_{v_i}[v_i] := RC_{v_i}[v_i] \bmod c(v_i)$

/* **Case 2-2:** assume $d(v_i)$ is assigned to $\hat{u} \in Ch(v_i)$. */
Initially $W_{Ch(v_i)} = \infty$, $RC_{Ch(v_i)} = 0$.
for every $u^* \in Ch(v_i)$
  $RC^*_{Ch(v_i)}[v_i] := \sum_{u \in Ch(v_i), W_{\downarrow}[u] \geq W_{\uparrow}[u]} (c(v_i) - (d(u) \bmod c(v_i)))$
  $W^*_{Ch(v_i)}[v_i] := \lceil d(v_i)/c(u^*) \rceil + \sum_{u \in Ch(v_i), W_{\downarrow}[u] < W_{\uparrow}[u]} W_{\downarrow}[u] + \sum_{u \in Ch(v_i), W_{\downarrow}[u] \geq W_{\uparrow}[u]} W_{\uparrow}[u] - \lfloor RC^*_{Ch(v_i)}[v_i]/c(v_i) \rfloor$
  $RC^*_{Ch(v_i)}[v_i] := RC^*_{Ch(v_i)}[v_i] \bmod c(v_i)$
  /* Consider $RC_{\uparrow}[u^*]$ or $RC_{\downarrow}[u^*]$ to reduce $W^*_{Ch(v_i)}[v_i]$. */
  if $W_{\downarrow}[u^*] \geq W_{\uparrow}[u^*]$
    $W^*_{Ch(v_i)}[v_i] := W^*_{Ch(v_i)}[v_i] - \lfloor (RC_{\uparrow}[u^*] + (c(u^*) - (d(v_i) \bmod c(u^*))))/c(u^*) \rfloor$
  else
    $W^*_{Ch(v_i)}[v_i] := W^*_{Ch(v_i)}[v_i] - \lfloor (RC_{\downarrow}[u^*] + (c(u^*) - (d(v_i) \bmod c(u^*))))/c(u^*) \rfloor$
  end if
  if $W^*_{Ch(v_i)}[v_i] < W_{Ch(v_i)}[v_i]$ or ($W^*_{Ch(v_i)}[v_i] = W_{Ch(v_i)}[v_i]$ and $RC^*_{Ch(v_i)}[v_i] > RC_{Ch(v_i)}[v_i]$)
    $W_{Ch(v_i)}[v_i] = W^*_{Ch(v_i)}[v_i]$, $RC_{Ch(v_i)}[v_i] = RC^*_{Ch(v_i)}[v_i]$, and $\hat{u} = u^*$
    if $W_{\downarrow}[\hat{u}] \geq W_{\uparrow}[\hat{u}]$
      $flag[\hat{u}] := 1$
    else
      $flag[\hat{u}] := 0$
    end if
  end if

  /* Consider the case $d(u^*)$ is not assigned upward if u^* is *undetermined*. */
  if $W_{\downarrow}[u^*] = W_{\uparrow}[u^*]$ and $RC_{\downarrow}[u^*] > RC_{\uparrow}[u^*]$
    $RC^*_{Ch(v_i)}[v_i] := \sum_{u \in Ch(v_i), u \neq u^*, W_{\downarrow}[u] \geq W_{\uparrow}[u]} (c(v_i) - (d(u) \bmod c(v_i)))$
    $W^*_{Ch(v_i)}[v_i] := \lceil d(v_i)/c(u^*) \rceil + \sum_{u \in Ch(v_i),\ W_{\downarrow}[u] < W_{\uparrow}[u]} W_{\downarrow}[u] + \sum_{u \in Ch(v_i),\ u \neq u^*,\ W_{\downarrow}[u] \geq W_{\uparrow}[u]} W_{\uparrow}[u] + W_{\downarrow}[u^*] - \lfloor RC^*_{Ch(v_i)}[v_i]/c(v_i) \rfloor$
    $RC^*_{Ch(v_i)}[v_i] := RC^*_{Ch(v_i)}[v_i] \bmod c(v_i)$
    /* Consider $RC_{\downarrow}[u^*]$ to reduce $W^*_{Ch(v_i)}[v_i]$. */
    $W^*_{Ch(v_i)}[v_i] := W^*_{Ch(v_i)}[v_i] - \lfloor (RC_{\downarrow}[u^*] + (c(u^*) - (d(v_i) \bmod c(u^*))))/c(u^*) \rfloor$
    if $W^*_{Ch(v_i)}[v_i] < W_{Ch(v_i)}[v_i]$ or ($W^*_{Ch(v_i)}[v_i] = W_{Ch(v_i)}[v_i]$ and

$RC^*_{Ch(v_i)}[v_i] > RC_{Ch(v_i)}[v_i]$)
$W_{Ch(v_i)}[v_i] = W^*_{Ch(v_i)}[v_i]$, $RC_{Ch(v_i)}[v_i] = RC^*_{Ch(v_i)}[v_i]$, $\hat{u} = u^*$, and
$flag[\hat{u}] := 0$
end if
end if
end for
if $W_{v_i}[v_i] < W_{Ch(v_i)}[v_i]$ or ($W_{v_i}[v_i] = W_{Ch(v_i)}[v_i]$ and $RC_{v_i}[v_i] \geq RC_{Ch(v_i)}[v_i]$)
$W_{\downarrow}[v_i] = W_{v_i}[v_i]$, $RC_{\downarrow}[v_i] = RC_{v_i}[v_i]$, $\hat{u} = v_i$
else
$W_{\downarrow}[v_i] = W_{Ch(v_i)}[v_i]$, $RC_{\downarrow}[v_i] = RC_{Ch(v_i)}[v_i]$
end if

/* determine the demand assignment and the pointer */
if $W_{\uparrow}[v_i] < W_{\downarrow}[v_i]$ or ($W_{\uparrow}[v_i] = W_{\downarrow}[v_i]$ and $RC_{\uparrow}[v_i] \geq RC_{\downarrow}[v_i]$)
/* v_i is determined to be assigned upward */
$\hat{u} = p(v_i)$, $f(v_i, p(v_i)) = d(v_i)$
else if $W_{\uparrow}[v_i] > W_{\downarrow}[v_i]$ /* v_i is determined to be assigned downward */
$f(v_i, \hat{u}) = d(v_i)$, assign $\hat{u}$ and all its undetermined descendants downward by pointers until $flag = 1$
else /* v_i is undetermined */
Set a pointer from v_i to $\hat{u}$
end if
for every *undetermined* $u^* \in Ch(v_i)$
if $u^* \neq \hat{u}$ or ($u^* = \hat{u}$ and $flag[\hat{u}] = 1$)
$f(u^*, v_i) = d(u^*)$ and every undetermined descendant of u^* is set to be assigned upward by pointers
end if
end for
end for
End

Algorithm Min_Capacitated_Domination_Splittable_demand_on_Trees MCDST
Input
1. A tree $T = (V, E)$ with a postorder tree traversal $v_1, v_2, \ldots, v_n$
2. A capacity function $c : V \to \mathbb{Z}^+ \cup \{0\}$
3. A demand function $d : V \to \mathbb{Z}^+ \cup \{0\}$

Output
The minimum cardinality of a capacitated dominating set D of T and the optimal demand assignment function $f : V \times D \to \mathbb{Z}^+ \cup \{0\}$, where $f(v, \hat{u})$ corresponds to each $\hat{u} \in D$, $\forall v \in V$.

Begin
for $i = 1$ to n
Let $Ch(v_i)$ be the child set of v_i and $p(v_i)$ the parent of v_i.
Let $Ch_d(v_i)$ and $Ch_u(v_i)$ be the determined and undetermined child sets of v_i, respectively.

/* exhaust the residue capacity of $Ch_d(v_i)$ */
for each $p_j \in Ch_d(v_i)$ **do**
if $RD[v_i] \leq RC[p_j]$ then
$f(v_i, p_j) = f(v_i, p_j) + RD[v_i]$

$RD[v_i] = 0$
break
end if
$f(v_i, p_j) = f(v_i, p_j) + RC[p_j]$
$RD[v_i] = RD[v_i] - RC[p_j]$
end for
/* perform the transformation to Relaxed Knapsack Problem */
$W = 0$
for each $q_k \in Ch_u(v_i)$ **do**
Let $a_k = c(q_k) - RD[q_k]$ and $b_k = c(v_i) - RD[q_k]$
$W = W + a_k$
end for
$W = W - RD[v_i]$

Use (a, b, W) as the Relaxed Knapsack Problem instance and solve it.
Let B^* be the solution and backtrack B^* to obtain the optimal set A.

/* determine the arrangement of the undetermined children */
for each $q_k \in Ch_u(v_i)$ **do**
if $k \in A$ **then** /* assign $RD[q_k]$ upward */
$f(q_k, v_i) = f(q_k, v_i) + RD[q_k]$
$RC[v_i] = RC[v_i] + c(v_i) - RD[q_k]$
else /* assign $RD[q_k]$ downward */
$f(q_k, q_k) = f(q_k, q_k) + RD[q_k]$
if $RD[v_i] < c(q_k) - RD[q_k]$ **then**
$f(v_i, q_k) = f(v_i, q_k) + RD[v_i]$
$RD[v_i] = 0$
else
$f(v_i, q_k) = f(v_i, q_k) + (c(q_k) - RD[q_k])$
$RD[v_i] = RD[v_i] - (c(q_k) - RD[q_k])$
end if
end if
end for
/* exhaust the residue capacity of v_i, $RC[v_i]$ */
if $RC[v_i] \geq RD[v_i] > 0$ **then**
$f(v_i, v_i) = f(v_i, v_i) + RD[v_i]$
$RC[v_i] = RC[v_i] - RD[v_i]$
$RD[v_i] = 0$
else
$f(v_i, v_i) = f(v_i, v_i) + RC[v_i]$
$RD[v_i] = RD[v_i] - RC[v_i]$
$RC[v_i] = 0$
end if
Let u^* be the vertex with maximum capacity in $N[v_i] = Ch(v_i) \cup \{v_i\} \cup \{p(v_i)\}$.
$f(v_i, u^*) = f(v_i, u^*) + \lfloor \frac{RD[v_i]}{c(u^*)} \rfloor \cdot c(u^*)$
$RD[v_i] = RD[v_i] \bmod c(u^*)$

/* check the undetermined condition of v_i */
if $0 < RD[v_i] \leq c(p(v_i)) < c(v_i)$ **then**
Mark v_i as *undetermined*

```
      else
        Mark v_i as determined

        if RD[v_i] > max{c(v_i), c(p(v_i))} then
          /* assign v_i to u* */
          f(v_i, u*) = f(v_i, u*) + RD[v_i]
        else if 0 < RD[v_i] ≤ c(p(v_i)) then
          /* assign v_i to p(v_i)
          f(v_i, p(v_i)) = f(v_i, p(v_i)) + RD[v_i]
          RC[p(v_i)] = c(p(v_i)) − RD[v_i]
        else if 0 < RD[v_i] then
          /* assign v_i to v_i itself
          f(v_i, v_i) = f(v_i, v_i) + RD[v_i]
          RC[v_i] = c(v_i) − RD[v_i]
        end if
        RD[v_i] = 0
      end if
    end for
  End
```

Algorithm Min_Capacitated_Domination_Approximation_on_Weighted_Graphs MCDAWG

Input

1. A graph $G = (V, E)$
2. A capacity function $c : V \to \mathbb{R}^+ \cup \{0\}$
3. A demand function $d : V \to \mathbb{R}^+ \cup \{0\}$
4. A cost function $w : V \to \mathbb{R}^+ \cup \{0\}$

Output

An feasible demand assignment function $f : V \times D \to \mathbb{R}^+ \cup \{0\}$ with Δ^*-approximation

Begin

$V^\phi := V$ [V^ϕ is the set of unassigned vertices].
$V_c := V$ [V_c is the set of closed vertices].
for every $v \in V$
  $D_N^\phi(v) := V^\phi \cap N[v]$. $d_N^\phi(v) := \sum_{u \in D_N^\phi(v)} d(u)$.
  $w^\phi(v) := w(v)$.
  $f(u, v) = 0, u \in N[v]$.
end for
while $V^\phi \neq \emptyset$
  $r_v := w^\phi(v) / \min\{c(v), d_N^\phi(v)\}, v \in V^\phi$.
  $u := \arg\min\{r_v : v \in V^\phi\}$. /* break ties arbitrarily */
  $V^\phi := V^\phi \backslash \{u\}$.
  $w^\phi(v) := w^\phi(v) - r_u \cdot \min\{c(v), d_N^\phi(v)\}, v \in V^\phi$.
  $f(p, u) = f(p, u) + d(p)$, for $p \in D_N^\phi(u)$.
  if $d_N^\phi(u) \leq c(u)$ /* reassignment operations */
    for every $v \in N[u]$ with $D_{v\bar{u}}^c \neq 0$
      $f(v, u) = f(v, u) + D_{v\bar{u}}^c$.
      while $D_{v\bar{u}}^c \neq 0$
      /* update $f(v, w)$, $w \in N[v]$, after the reassignment */

```
            Find a vertex w ∈ N[v] such that w ≠ v and f(v, w) ≠ 0.
            if f(v, w) > D^c_{vū}
              f(v, w) = f(v, w) − D^c_{vū}.
              D^c_{vū} = 0.
            else
              D^c_{vū} = D^c_{vū} − f(v, w).
              f(v, w) = 0.
            end if
          end while
        end for
      end if
      for every p ∈ D^φ_N(u)
      /* update D^c_{qp̄}, q ∈ D^φ_N(p), ∀p ∈ D^φ_N(u), after the reassignment */
        V^φ := V^φ\{p}, D^φ_N(p) := D^φ_N(p)\{u}.
        for every q ∈ D^φ_N(p)
          D^φ_N(q) = D^φ_N(q)\{p}.
          if d^φ_N(q) ≤ c(q)
            D^c_{qp̄} = D^c_{qp̄} + d(p).
          else if d^φ_N(q) − d(p) ≤ c(q)
            D^c_{qp̄} = D^c_{qp̄} + d(p) + c(q) − d^φ_N(q).
          end if
          d^φ_N(q) = d^φ_N(q) − d(p).
        end for
      end for
    end while
End
```

References

1. Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., Pandit, V.: Local search heuristics for k-median and facility location problems. SIAM J. Comput. **33**(3), 544–562 (2004)
2. Bar-Yehuda, R., Even, S.: A linear-time approximation algorithm for the weighted vertex cover problem. J. Algorithms **2**, 198–203 (1981)
3. Bar-Ilan, J., Kortsarz, G., Peleg, D.: Generalized submodular cover problems and applications. Theor. Comput. Sci. **250**, 179–200 (2001)
4. Bartal, Y., Charikar, M., Raz, D.: Approximating min-sum k-clustering in metric spaces. In: Proceedings of 33th ACM Symposium on Theory of Computing, pp. 11–20 (2001)
5. Charikar, M., Guha, S.: Improved combinatorial algorithms for facility location and k-median problems. In: Proceedings of 40th IEEE Symposium of Foundations of Computer Science, pp. 378–388 (1999)
6. Charikar, M., Guha, S., Tardos, É., Shmoys, D.B.: A constant-factor approximation algorithm for the k-median problem. J. Comput. Syst. Sci. **65**(1), 129–149 (2002)
7. Cheng, X., Huang, X., Li, D., Wu, W., Du, D.-Z.: A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. Networks **42**(4), 202–208 (2003)
8. Chuzhoy, J., Rabani, Y.: Approximating k-median with non-uniform capacities. In: Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms, pp. 952–958 (2005)
9. Chuzhoy, J., Naor, J.S.: Covering problems with hard capacities. SIAM J. Comput. **36**(2), 498–515 (2006)

10. Chudak, F.A., Shmoys, D.B.: Improved approximation algorithms for a capacitated facility location problem. In: Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms, pp. 875–876 (1999)
11. Chudak, F.A., Williamson, D.P.: Improved approximation algorithms for capacitated facility location problems. Math. Program. **102**(2), 207–222 (2005)
12. Chvátal, V.: A greedy heuristic for the set-covering problem. Math. Oper. Res. **4**(3), 233–235 (1979)
13. Feige, U.: A threshold of ln n for approximating set cover. J. ACM **45**(4), 634–652 (1998)
14. Gandhi, R., Halperin, E., Khuller, S., Kortsarz, G., Srinivasan, A.: An improved approximation algorithm for vertex cover with hard capacities. J. Comput. Syst. Sci. **72**(1), 16–33 (2006)
15. Gandhi, R., Khuller, S., Parthasarathy, S., Srinivasan, A.: Dependent rounding and its applications to approximation algorithms. J. ACM **53**(3), 324–360 (2006)
16. Guha, S., Hassin, R., Khuller, S., Or, E.: Capacitated vertex covering. J. Algorithms **48**(1), 257–270 (2003)
17. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: Domination in Graphs: The Theory. Dekker, New York (1998)
18. Hochbaum, D.S.: Approximation algorithms for the set covering and vertex cover problems. SIAM J. Comput. **11**(3), 555–556 (1982)
19. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. J. ACM **22**(4), 463–468 (1975)
20. Jain, K., Vazirani, V.V.: Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. J. ACM **48**(2), 274–296 (2001)
21. Jain, K., Mahdian, M., Saberi, A.: A new greedy approach for facility location problems. In: Proceedings of 34th ACM Symposium on Theory of Computing, pp. 731–740 (2002)
22. Johnson, D.S.: Approximation algorithms for combinatorial problems. J. Comput. Syst. Sci. **9**(3), 256–278 (1974)
23. Korupolu, M., Plaxton, C., Rajaraman, R.: Analysis of a local search heuristic for facility location problems. J. Algorithms **37**(1), 146–188 (2000)
24. Levi, R., Shmoys, D.B., Swamy, C.: LP-based approximation algorithms for capacitated facility location. In: Proceedings of 10th Conference on Integer Programming and Combinatorial Optimization, pp. 206–218 (2004)
25. Liao, C.S., Chang, G.J.: Algorithmic aspect of k-tuple domination in graphs. Taiwan. J. Math. **6**, 415–420 (2002)
26. Liao, C.S., Chang, G.J.: k-tuple domination in graphs. Inf. Process. Lett. **87**(1), 45–50 (2003)
27. Lovász, L.: On the ratio of optimal and fractional covers. Discrete Math. **13**, 383–390 (1975)
28. Mahdian, M., Pál, M.: Universal facility location. In: Proceedings of 11th European Symposium on Algorithms, pp. 409–421 (2003)
29. Mahdian, M., Ye, Y., Zhang, J.: Approximation algorithms for metric facility location problems. SIAM J. Comput. **36**(2), 411–432 (2006)
30. Pál, M., Tardos, É., Wexler, T.: Facility location with nonuniform hard capacities. In: Proceedings of 42th Symposium of Foundations of Computer Science, pp. 329–338 (2001)
31. Shmoys, D.B., Tardos, É., Aardal, K.: Approximation algorithms for facility location problems. In: Proceedings of 29th ACM Symposium on Theory of Computing, pp. 265–274 (1997)
32. Zhang, J., Chen, B., Ye, Y.: A multi-exchange local search algorithm for the capacitated facility location problem. Math. Oper. Res. **30**(2), 389–403 (2005)