國立臺灣大學電機資訊學院資訊工程學研究所

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

有容量的支配集問題的近似演算法

Approximation Algorithms for the Capacitated Domination Problem

陳翰霖

Han-Lin Chen

指導教授：李德財 博士

Advisor: Der-Tsai Lee, Ph.D.

中華民國 99 年 7 月

July, 2010

中文摘要

我們考慮容量支配集問題。此問題模型化了一般的服務-需求問題並且是傳統的支配集問題的一個一般化。在這個問題裡，給定一圖並在各個 點上附有三個參數：價格、容量與需求。我們將要尋找一個最低價格的配置並滿足需求與容量上的限制。 我們在不同的需求配置模型上提供了多項式時間的近似演算法，此演算法也逼近了傳統的支配集問題的成果。 與已知的前人的結果結合起來，已經從兩個方面逼近了傳統支配集問題的成果。

Abstract

We consider the *Capacitated Domination* problem, which models a service-requirement assignment scenario and is also a generalization of the well-known *Dominating Set* problem. In this problem, given a graph with three parameters defined on each vertex, namely cost, capacity, and demand, we want to find an assignment of demands to vertices of least cost such that the demand of each vertex is satisfied subject to the capacity constraint of each vertex providing the service.

In terms of polynomial time approximations, we present logarithmic approximation algorithms with respect to different demand assignment models for this problem on general graphs, which also establishes the corresponding approximating results to the well-known approximations of the traditional *Dominating Set* problem. Together with previously known results, this closes the problem of generally approximating the optimal solution.

Contents

List of Figures

Chapter 1

Introduction

For decades, the *domination* problem has been one of the most fundamental and well-known problems in both graph theory and combinatorial optimization. Given a graph $G = (V, E)$ and an integer k, the *domination* problem asks for a subset $D \subseteq V$ whose cardinality does not exceed k such that every vertex in the graph either belongs to this set or has a neighbor which does. As this problem is known to be NP-hard, approximation algorithms have been proposed in the literature. On one hand, a simple greedy algorithm is shown to achieve a guaranteed ratio of $O(\ln n)$ [7, 15, 19], where n is the number of vertices, which is later proven to be the approximation threshold by Feige [9]. On the other hand, algorithms based on dual-fitting provide a guaranteed ratio of Δ [14], where Δ is the degree of the graph[1].

Besides the *domination* problem itself, a vast body of work has been proposed in the literature, considering possible variations from purely theoretical aspects to practical applications. See [13, 20] for a detailed survey. In particular, variations of the *domination* problem occur in numerous practical settings, ranging from strategic decisions, such as locating radar stations or emergency services, to computational biology and to voting systems. For example, Haynes et al. [12] considered *Power Domination Problems* in electric networks [12, 18] while Wan et al. [21] considered *Connected Domination Problems* in wireless ad hoc networks.

[1]The degree of a graph is defined to be the maximum degree of the vertices of the graph.

Motivated by a general service-requirement assignment model, Kao et al. [16] considered a generalized domination problem called *capacitated domination*. In this problem, the input graph is given with tri-weighted vertices, referred to as *cost*, *capacity*, and *demand*, respectively. The demand of a vertex stands for the amount of service it requires from its adjacent vertices (including itself) while the capacity of a vertex represents the amount of service it can provide when it's selected as a server. The goal of this problem is to find a dominating multi-set as well as a demand assignment function such that the overall cost of the multi-set is minimized. According to different underlying applications, there are two different demand assignment models, namely *splittable* demand model and *unsplittable* demand model, depending on whether or not the demand of a vertex is allowed to be served by different vertices. Moreover, there has been work studying the variation when the number of copies, or *multiplicity* of each vertex in the dominating multi-set, is limited, referred to as *hard* capacity, and as *soft* capacity when no such limit is specified. Kao et al. [16] considered the soft capacitated domination problem with splittable demand and provided a $(\Delta + 1)$-approximation for general graphs, where Δ is the degree of the graph. For special graph classes, they proved that even when the input graph is restricted to a tree, the soft capacitated domination problem with splittable demand remains NP-hard, for which they also provided a polynomial time approximation scheme.

This thesis intended to connect the investigation of the generalization of the domination problem to the classical results. If we consider the domination problem as a service-requirement model, the setting will be too naive to represent the real world problem such as placing wireless network base stations or constructing cellphone services.

In this thesis, we consider the (soft) *capacitated domination* problem and present logarithmic approximation algorithms with respect to different demand assignment models on general graphs. Specifically, we provide a $(\ln n)$-approximation for weighted unsplittable demand model, a $(4 \ln n + 2)$-approximation for weighted splittable demand model, and a $(2 \ln n + 1)$-approximation for unweighted splittable demand model, where n is the number of vertices. Together with the $(\Delta + 1)$-approximation result given by Kao et al. [16], this establishes a corresponding near-optimal approximation result to the original *domination* problem. Although the result may look natural, the greedy choice we make is not obvious when non-uniform capacity as well as non-uniform demand is taken into consideration.

Our algorithms are based on the greedy approach of Johnson et al. [7, 15, 19] in the sense that we keep choosing a vertex with the best efficiency in each iteration until the whole graph is dominated. The question is how we can cope with different demand assignment models to achieve the same guaranteed ratio $O(\ln n)$. In this work, we generalize the concept of efficiency and use it as our greedy choice. We describe the results in more detail in the following chapters.

Chapter 2

Preliminary

We assume that all the graphs considered in this thesis are simple and undirected. Let $G = (V, E)$ be a graph with vertex set V and edge set E. A vertex $v \in V$ is said to be adjacent to a vertex $u \in V$ if $(u, v) \in E$. The set of neighbors of a vertex $v \in V$ is denoted by $N_G(v) = \{u : (u, v) \in E\}$. The closed neighborhood of $v \in V$ is denoted by $N_G[v] = N_G(v) \cup \{v\}$. The subscript G in $N_G[v]$ will be omitted when there is no confusion.

2.1 Traditional Domination Problem

We begin by introducing the traditional domination problem, whose preliminary study serves as a basic step to a fundamental greedy technique and is very useful throughout this thesis. A vertex set D is said to be a dominating set if for all vertex v in G, $N_G[v] \cap D \neq \phi$. Given a graph, the domination problem asks for a dominating set of minimum cardinality. A greedy algorithm as well as the concept of the most efficient choice is provided by Johnson et al [7, 15, 19]. In their algorithm they maintain a vertex set U which contains exactly the vertices which are not dominated yet, and iteratively remove the vertices which are newly dominated during each iteration from U. Initially, set U as V. During the iteration the efficiency of a vertex v is defined to be the cardinality of $N_G[v] \cap U$. The proposed algorithm is to iteratively pick the vertex with the most efficiency, say u, and remove the vertices in $U \cap N_G[u]$ from U until U is empty. Figure 2.1 presents a high

```
ALGORITHM Log-Approx-Dominating
1: $U \longleftarrow V$
2: $D \longleftarrow \phi$
3: while $U \neq \phi$ do
4:     Pick a vertex, say $u$ in $V$ with the most efficiency, i.e., $|N_G[u] \cap U|$ is maximum, and put it into $D$.
5:     For each $e \in N_G[u] \cap U$ Set $price(e) = \frac{1}{|N_G[u] \cap U|}$
6:     Remove vertices in $N_G[u] \cap U$ out of $U$
7: end while
8: Compute the cardinality of $D$ and return it as the result.
```

Figure 2.1: The pseudo-code of the algorithm for the domination problem

level description of the algorithm.

The greedy strategy applies naturally to the domination problem. Although the algorithm is rather simple, the idea of picking the vertex with the most efficiency plays an important role in the analysis. In iteration j, let OPT_j be the cardinality of the optimal solution for the remaining problem instance. Clearly, it should be bounded above by OPT, the cardinality of the optimal solution of the original problem instance. Let $e_1, \ldots, e_n$ be the vertices in the chronological order of being dominated during the execution of the algorithm. We have following lemma.

Lemma 1. For each $k \in 1, \ldots, n$, $price(e_k) \leq \frac{OPT_j}{n-k+1}$, where e_k is dominated in iteration j

Proof. In the iteration in which element e_k was dominated, $n - |U|$ contains at least $n - k + 1$ elements. In iteration j the optimal solution for the remaining instance, OPT_j, must contain a vertex v which gives the maximum cardinality, $|N_G[v] \cap U|$. Since we always choose the vertex with maximum efficiency, that is, the vertex with the largest number of vertices in U which are adjacent to the selected vertex, the efficiency is no less than that of the vertex chosen in OPT_j. Thus we have $\frac{1}{price(e_k)} \leq |N_G[v] \cap U|$. Therefore we have $price(e_k) \leq \frac{OPT_j}{n-|U|} \leq \frac{OPT_j}{n-k+1}$ and the lemma follows. □

Theorem 2. Algorithm *Log-Approx-Dominating computes a* $(\ln n)$*-approximation for dominatig set problem, where* n *is the number of vertices.*

Proof. Since the cardinality of each vertex picked is distributed among the new vertices dominated, the total cardinality of D is equal to $\sum_{k=1}^{n} price(e_k)$. By Lemma 1 $\sum_{k=1}^{n} price(e_k) \leq \sum_{k=1}^{n} \frac{OPT_j}{n-k+1} \leq \sum_{k=1}^{n} \frac{OPT}{n-k+1} \leq \ln n \cdot OPT$ □

In the study of the domination problem, one may notice that the concept of the most efficient choice is an important idea to obtain a $\ln n$-approximation. In fact, the concept of efficiency is also a key to solving the approximation of the capacitated domination problem, which is a generalization of the domination problem.

2.2 Capacitated Domination Problem

In this problem we are given a graph $G = (V, E)$ with tri-weighted vertices, referred to as the cost, the capacity, and the demand of each vertex $u \in V$, denoted by $w(u)$, $c(u)$, and $d(u)$, respectively. The demand of a vertex stands for the amount of service it requires from its closed neighbourhood while the capacity of a vertex represents the amount of service it can provide each time when it's selected as a server.

Let D denote a multi-set of vertices of V and for any vertex $u \in V$, let $x_D(u)$ denote the *multiplicity* of u or the number of times of u in D. The cost of D, denoted $w(D)$, is defined to be $w(D) = \sum_{u \in D} w(u) \cdot x_D(u)$. The goal of the problem is to find a dominating multi-set and a demand assignment function such that the cost of the multi-set, $w(D)$, is minimized.

The demand assignment function $f : V \times V \rightarrow R^+ \cup 0$ is a function indicating the demand assignment in a solution. In particular $f(v, u)$ denotes the amount of demand of each vertex $v \in V$ which is assigned to a vertex $u \in N[v]$. Note that $f(v, u) = 0$ if $(u, v) \notin E$

Definition 1 (Capacitated Dominating Set)**.** A vertex multi-subset D is said to be a feasible capacitated dominating set with respect to a demand assignment function f if the following conditions hold.

- **Demand constraint:**

  $\sum_{u \in N_G[v]} f(v,u) \geq d(v)$, for each $v \in V$.

- **Capacity constraint:**

  $\sum_{u \in N_G[v]} f(u,v) \leq c(v) \cdot x_D(v)$, for each $v \in V$.

Given a problem instance, the capacitated domination problem asks for a capacitated dominating multi-set D and demand assignment function f such that $w(D)$ is minimized. In unsplittable demand model we require that $f(u,v)$ is either 0 or $d(u)$ for each edge $(u,v) \in E$. Note that since it is already NP-hard[1] to compute a feasible demand assignment function from a given feasible capacitated dominating multi-set when the demand cannot be split, it is natural to require the demand assignment function to be specified, in addition to the optimal vertex multi-set itself.

To demonstrate the difference between the solutions of the domination problem and capacitated domination problem, here are Figure 2.2 and Figure 2.3 which show the solutions for one of its instances respectively.

[1]This is easily verified by making a reduction from simple combinatorial optimization problems such as SUBSET SUM.

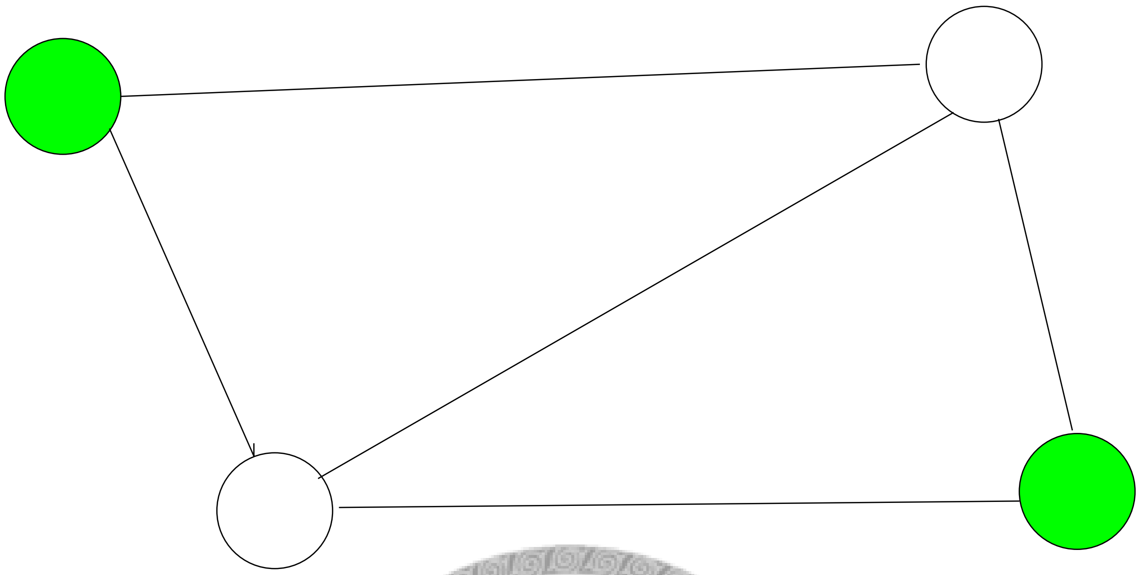


Figure 2.2: An optimal solution of the domination problem.

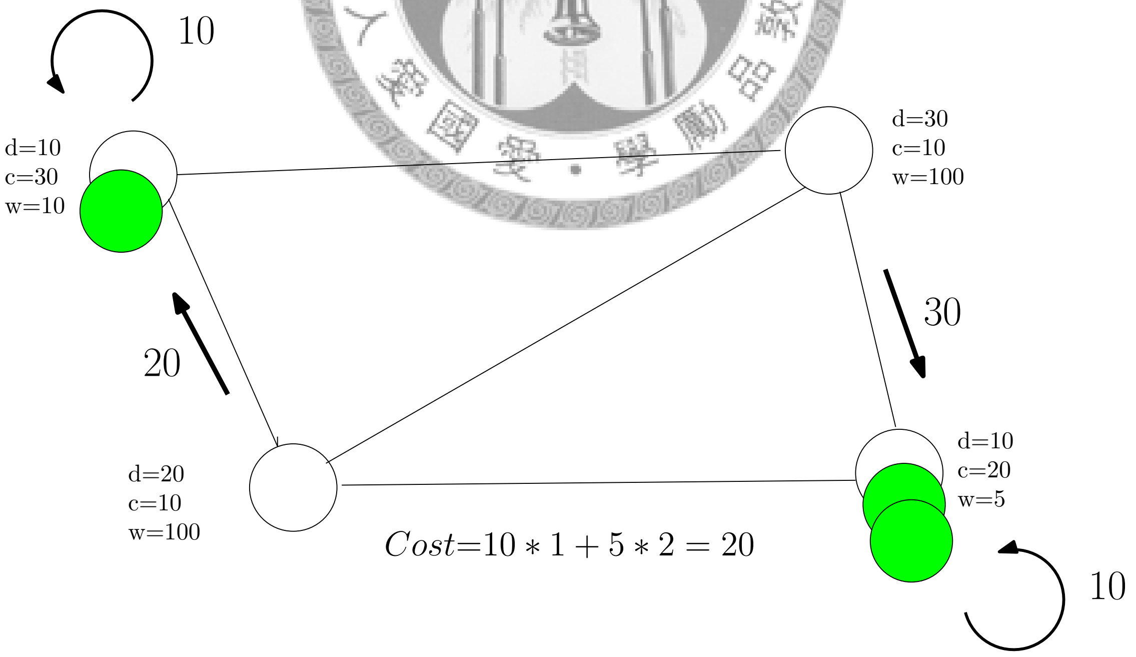


Figure 2.3: An optimal solution of capacitated domination problem.

Chapter 3

Weighted Unsplittable Demand

In this chapter, we consider the *weighted capacitated domination problem with unsplittable demand* and provide a simple greedy algorithm that achieves the approximation guarantee of $\ln n$. First, we recall our earlier definition of unsplittable model and describe the difference of demand assignments between the traditional domination problem and the capacitated domination problem with unsplittable model, which leads to a new definition of efficiency. Second, we give a greedy algorithm. Finally, we will discuss the approximation guarantee and the time complexity of the greedy algorithm.

Let us start with the difference between feasible solutions of traditional domination problem and capacitated domination problem. To take an example, please refer to Figure 2.2 and Figure 2.3. For the domination problem once we pick a vertex, we are able to dominate all of vertices in its neighborhood. For capacitated domination problem, on the other hand, we have to consider the capacity and the demand assigning to it. Sometimes we have to take multiple copies of a vertex to satisfy the demand assigning to it, like the bottom right vertex in Figure 2.3. This main difference leads to a problem that we cannot consider the most efficient choice only about taking one vertex, but multiple copies of a vertex. And moreover, we have to consider the demand assigned to the vertex.

3.1 The Definition of Efficiency

To explain our idea of the most efficient choice, let us, for the moment, consider a simple question: if we need to dominate i neighbors of a vertex, say u, by taking some copies of u, how should we choose the neighbors to be dominated such that the number of copies we need is as few as possible. It's obvious that no matter how much capacity u has, the best choice is always to select the vertex that dominates the i neighbors with the first i smallest demands. This simple observation indicates an efficient way of dominating i neighbors of a vertex u. There is one further thing that this also indicates the way of demand assignment because those i neighbors have been identified.

It seems that we can conclude a way of choosing the most cost-efficient demand assignment together with some copies of one vertex. However, there is another problem. As shown in Figure 3.1 and Figure 3.2 the number of neighbors to be dominated will affect the cost-efficiency of the demand assignment, which means we have to compute the cost-efficiency for all possible numbers of neighbors to dominate, when we determine the best demand assignment. Now we can formally define the efficiency of a vertex in capacitated domination problem for unsplittable model.

Given a graph instance, let U be the set of vertices which are not dominated yet. For each vertex $u \in V$, let $N_{ud}[u] = U \cap N[u]$ be the set of undominated vertices in the closed neighborhood of u. Without loss of generality, we shall assume that the elements of $N_{ud}[u]$, denoted by $v_{u,1}, v_{u,2}, \ldots, v_{u,|N_{ud}[u]|}$, are sorted in non-decreasing order of their demands in the remaining chapter.

Here we define the efficiency of a vertex u as

$$\max_{1 \le i \le |N_{ud}[u]|} \frac{i}{w(u) \cdot x_u(i)},$$

where

$$x_u(i) = \left\lceil \frac{\sum_{1 \le j \le i} d(v_{u,j})}{c(u)} \right\rceil$$

is the number of copies of u selected in order to dominate $v_{u,1}, v_{u,2}, \ldots,$ and $v_{u,i}$. Note that this efficiency not only indicates a number but also a demand assignment with regard

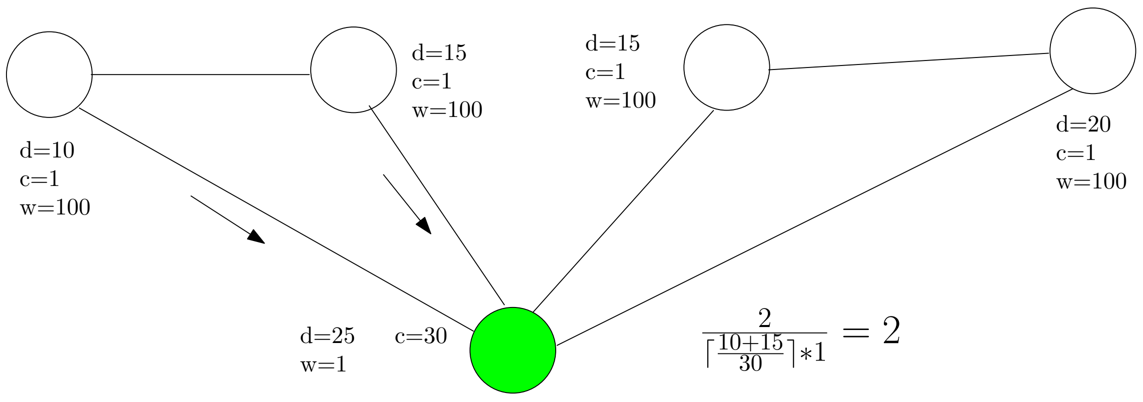


Figure 3.1: The cost-efficiency of a demand assignment with 2 neighbors

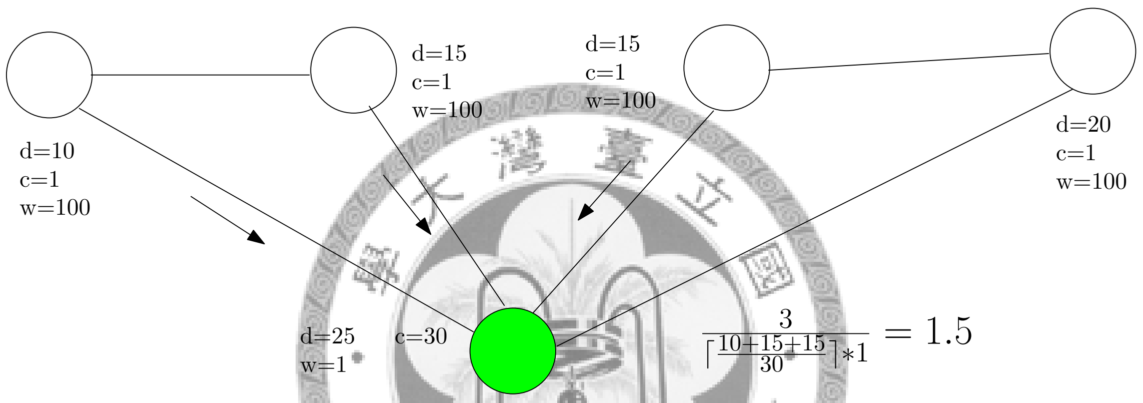


Figure 3.2: The cost-efficiency of a demand assignment with 3 neighbors

to u.

3.2 The Algorithm for Capacitated Domination Problem with Unsplittable Model

After introducing the definition of efficiency, we are now in a position to present the algorithm which originates and differs slightly from an algorithm proposed by Johnson et al.[7, 15, 19] for the capacitated domination problem with unsplittable model. In each iteration, the algorithm chooses a vertex of the most efficiency from V of the remaining graph, say u, which accompanies a demand assignment as well as a number of copies of u. We repeatedly find the most efficient vertex and assign the demands until all the

```
ALGORITHM Unsplit-Log-Approx
1: $U \longleftarrow V$
2: while $U \neq \phi$ do
3:     Pick a vertex in $V$ with the most efficiency, say $u$.
4:     let $k = \arg\max_{1 \le i \le |N_{ud}[u]|} \frac{i}{w(u) \cdot x_u(i)}$.
5:     Assign the demand of each vertex in $\{v_{u,1}, v_{u,2}, \ldots, v_{u,k}\}$ to $u$ and remove them
       from $U$.
6: end while
7: compute from the assignment the weight of the dominating set, and return the result.
```

Figure 3.3: The pseudo-code for the weighted unsplittable demand model.

demands of each vertex of the graph have been assigned. A high-level description of this algorithm is presented in Figure 3.3. In Figure 3.4 we demonstrate an example with the result computed by our algorithm.

3.3 Analysis

Now we shall discuss the approximation ratio achieved by our algorithm. Using the definition of efficiency we can conclude a lemma very similar to Lemma 1. In iteration j, let OPT_j be the cost of the optimal solution for the remaining problem instance, which is clearly upper bounded by the cost, OPT, of the optimal solution for the input instance. During the algorithm, we say a vertex is undominated if its demand has not been assigned yet. Let the number of undominated vertices at the beginning of iteration j be n_j, and the number of vertices that are newly dominated in iteration j be k_j.

Denote by S_j the cost in iteration j. Note that $S_j = w(u) \cdot x_D(u)$, where u is the most efficient vertex chosen in iteration j. Assume that the algorithm repeats for m iterations. We have the following lemma.

Lemma 3. For each j, $1 \le j \le m$, we have $S_j \le \frac{k_j}{n_j} \cdot OPT_j$.

Proof. Let's consider OPT_j. OPT_j can be expressed as $\sum_{v\in V} w(v)\cdot x_D(v)$, where $x_D(v)$ is the number of copies of v in OPT_j. Then we have following equation:

$$\frac{n_j}{OPT_j} = \frac{K(v_1)+K(v_2)+\cdots+K(v_n)}{x_D(v_1)+x_D(v_2)+\cdots+x_D(v_n)},$$

where $K(v)$ denotes the number of vertices assigned to v. Thus there must exist a v_i such that $\frac{K(v_i)}{x_D(v_i)} \geq \frac{n_j}{OPT_j}$.

Since we always choose the vertex with the maximum efficiency, the efficiency is no less than that of each vertex chosen in OPT_j, which is no less than $\frac{K(v_i)}{x_D(v_i)}$. Therefore we have $\frac{k_j}{S_j} \geq \frac{K(v_i)}{x_D(v_i)} \geq \frac{n_j}{OPT_j}$ and the lemma follows. □

Lemma 4. Algorithm *Unsplit-Log-Approx* returns a $(\ln n)$-approximation for weighted capacitated domination problem with unsplittable demands, where n is the number of vertices.

Proof. To see that the algorithm produces a logarithmic approximation, take the sum over each S_j, $1 \leq j \leq m$ and observe that $n_{j+1} = n_j - k_j$, we have

$$\sum_{1\leq j\leq m} S_j \leq \sum_{1\leq j\leq m} \frac{k_j}{n_j}\cdot OPT_j$$

$$\leq \left(\sum_{1\leq j\leq n} \frac{1}{j}\right)\cdot OPT \leq \ln n \cdot OPT$$

□

For the time complexity of the algorithm, in each iteration it takes $O(n^2)$ time to find the most efficient vertex, and since it repeats for at most n iterations, the time complexity is in $O(n^3)$. We conclude this section with the following theorem.

Theorem 5. Algorithm *Unsplit-Log-Approx* computes a $(\ln n)$-approximation for weighted capacitated domination problem with unsplittable demands in $O(n^3)$ time, where n is the number of vertices.

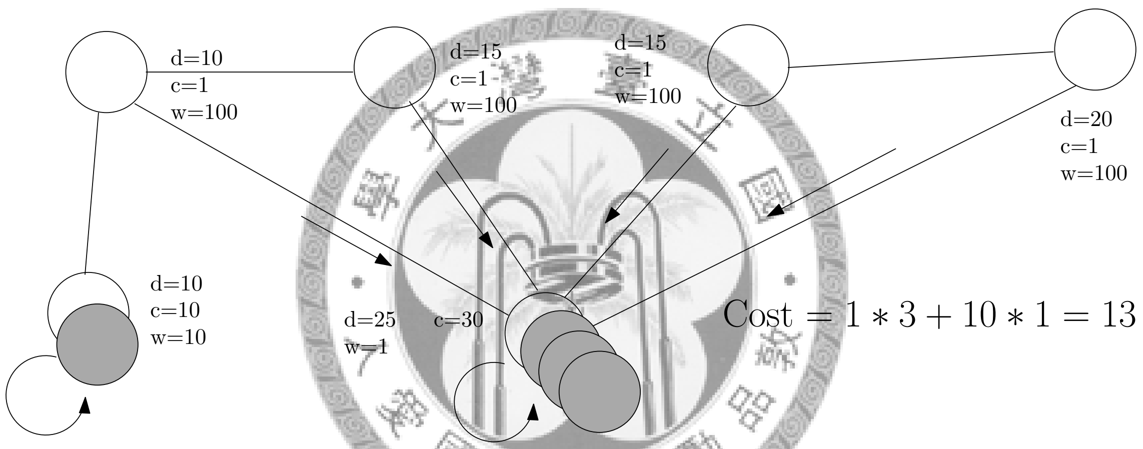


Figure 3.4: A feasible solution computed by our algorithm.

Chapter 4

Weighted Splittable Demand

In this chapter, we present an algorithm that produces a $(4 \ln n + 2)$-approximation for the *weighted capacitated domination problem with splittable demand*. Considering the major difference between unsplittable model and splittable model, which is the way the demand is assigned, we first illustrate the new concept of domination. Second, we modify the definition of efficiency to cope with this model. Then we present a modified greedy algorithm to achieve the approximation guarantee. Finally, we discuss the time complexity of our algorithm.

4.1 The Difference of Demand Assignment between Unsplittable Model and Splittable Model

The difference between this model and the previous one is that we can partially assign the demand of a vertex. Figure 4.1 shows an example. In Figure 4.1 we can see a vertex whose demand 15 gets partially assigned to the vertex in the bottom. In unspltable model if we want to assign the demand of a vertex, we must assign the entire demand. In that case we can regard a demand assignment as a domination of the whole vertex. However in splittable model since we can partially assign the demand of a vertex, we need a more general concept of satisfying the demands of vertices. Let's consider a demand assignment of assigning $f(v_1, u), f(v_2, u), \cdots, f(v_k, u)$ of demand from $v_1, v_2, \cdots v_k$ to a vertex u. We say that u dominates $\sum_{1 \le i \le k} \frac{f(v_i, u)}{d(v_i)}$. Let's take Figure 4.1 as an example again. In this demand assignment the vertex dominates $\frac{10}{10} + \frac{15}{15} + \frac{5}{15} = 2\frac{1}{3}$ number of vertices.

4.2 Definition of Efficiency

In the sense of handling demand assignment, we can define the efficiency of a vertex for splittable model now. Given a graph instance the demand of a vertex may be partially served. The unsatisfied portion of the demand is called *residue demand*. For each vertex $u \in V$, let $\mathrm{rd}(u)$ be the residue demand of u. $\mathrm{rd}(u)$ is set equal to $d(u)$ initially, and will be updated accordingly when certain portion of the residue demand is assigned. u is said to be completely satisfied when $\mathrm{rd}(u) = 0$.

We also inherit the notation used in the previous chapter. We assume that the elements of $N_{ud}[u]$, written as $v_{u,1}, v_{u,2}, \ldots, v_{u,|N_{ud}[u]|}$, are sorted according to their demands in non-decreasing order.

For each vertex $u \in V$, let j_u with $0 \le j_u \le |N_{ud}[u]|$ be the maximum index such that $c(u) \ge \sum_{i=1}^{j_u} \mathrm{rd}(v_{u,i})$. Let $X(u) = \sum_{i=1}^{j_u} \mathrm{rd}(v_{u,i})/d(v_{u,i})$ be the sum of the effectiveness over the vertices whose residue demand could be completely served by a single copy of u. In addition, we let $Y(u) = (c(u) - \sum_{i=1}^{j_u} \mathrm{rd}(v_{u,i}))/d(v_{u,j_u+1})$ if $j_u < |N_{ud}[u]|$ and $Y(u) = 0$ otherwise. The efficiency of u is defined as $(X(u) + Y(u))/w(u)$.

The definition of efficiency for splittable model here seems complicated, but the idea is rather simple. If we pick a copy of a vertex its capacity will be exhausted by the residue demand of vertices in its neighborhood, as in Section 3.1, in the order of their demand. It's interesting to note that we only consider one copy of a vertex here. Since we can dominate a portion of demand of a vertex in this model, taking one more copy wouldn't increase the efficiency.

It may be worth pointing out here that the reason we need to modify our algorithm for splittable model is that the concept of ”the number of undominated vertices” is different. In the previous chapter, we define n_j as the number of undominated vertices, but here it

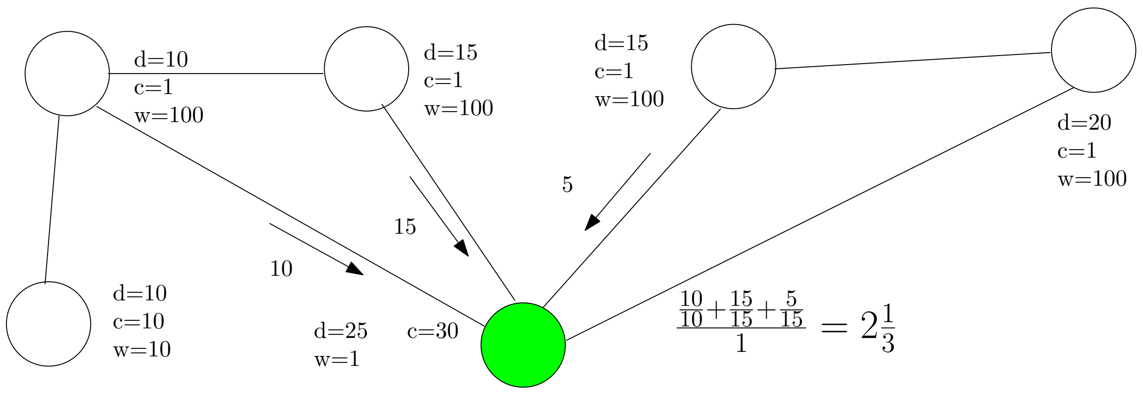


Figure 4.1: A partially demand assignment in splittable model.

would be $\sum_{v \in V} \frac{\mathrm{rd}(v)}{d(v)}$. This will cause some problems in the analysis, as can be seen in Theorem 4. Since n_j may no longer be integral now, we can't arrange the sum of the costs into a harmonic series. As a result we will need to develop another greedy approach in the next section to copy with this problem.

4.3 The Algorithm for Capacitated Domination Problem with Splittable Model

In our algorithm for capacitated domination problem for splittable model, we will also perform a greedy approach. That is, in each iteration the algorithm chooses the vertex of the most efficiency from V, where the efficiency is defined as in the previous section. In addition, we will take care of the following two cases. First, if the demand assignment indicated by the vertex of the most efficiency is unable to satisfy the residue demand of a single vertex, we will replace the demand assignment by the following: If the vertex of the most efficiency is u, we will assign $c(u) \cdot \left\lfloor \frac{\mathrm{rd}(v_{u,1})}{c(u)} \right\rfloor$ to u by taking $\left\lfloor \frac{\mathrm{rd}(v_{u,1})}{c(u)} \right\rfloor$ copies of u. See Figure 4.2 and Figure 4.3 as an example. One can easily see that this replaced demand assignment has the same efficiency. The reason we replace the demand assignment is because of the analysis of approximation radio which will be explained in details in the next section.

Second, in each iteration after we assign demand to the vertex of the most efficiency,

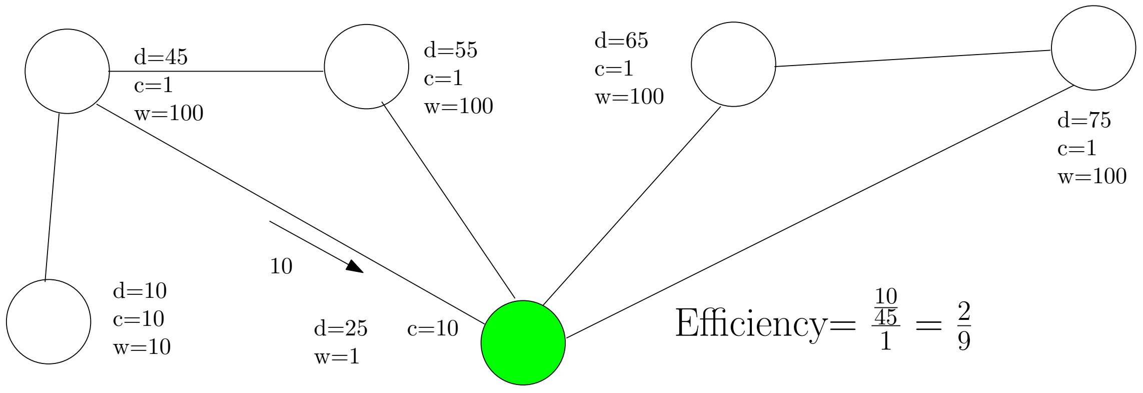
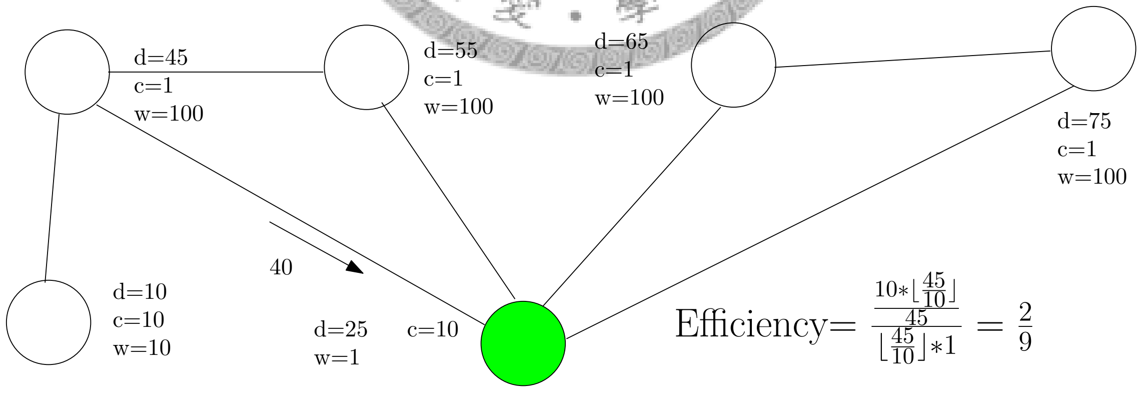


Figure 4.2: The demand assignment to be replaced



d=45
c=1
w=100
d=55
c=1
w=100
d=65
c=1
w=100
d=75
c=1
w=100
40
d=10
c=10
w=10
d=25
w=1
c=10
Efficiency= $\frac{\frac{10*\lfloor\frac{45}{10}\rfloor}{45}}{\lfloor\frac{45}{10}\rfloor*1} = \frac{2}{9}$

Figure 4.3: The demand assignment to replace

we will search for the whole graph. If there exists a vertex u whose residue demand is below half of its original demand, i.e., $0 < \mathrm{rd}(u) < \frac{1}{2} \cdot d(u)$, the algorithm immediately doubles the demand assignment of u to the set of vertices partially serving u. To do this, during the algorithm we will maintain for each vertex $u \in V$ a set of vertices, denoted by $\mathrm{map}(u)$, which consists of vertices that have been assigned partial demand of u before u is completely satisfied. That is, for each $v \in \mathrm{map}(u)$ we have a non-zero demand assignment of u to v. Note that in this way, we can completely satisfy the demand of u. The reason is that since u's residue demand is below half of its original demand the assignment in $\mathrm{map}(u)$ must have assigned at least half of u's demand, i.e., $\sum_{v \in \mathrm{map}(u)} f(u, v) > \frac{1}{2} \cdot d(u)$, doubling the assignment in $\mathrm{map}(u)$ will certainly completely satisfy the residue demand of u . A high-level description of this algorithm is presented in Figure 4.4 and Figure 4.5 shows a solution computed by our algorithm.

4.4 Analysis

In this section we will analyze the approximation ratio and the time complexity of our algorithm. The modified algorithm provides us some good properties to achieve the aforementioned approximation ratio. Together with the processing of the two cases mentioned above, we have the following observation.

Observation 1. After each iteration, the residue demand of each unsatisfied vertex is at least half of its original demand.

Clearly, the observation holds in the beginning when the demand of each vertex is not yet assigned. For later stages, we argue that the algorithm properly maintains map so that in our second greedy choice, whenever there exists a vertex u for which $0 < \mathrm{rd}(u) < \frac{1}{2} \cdot d(u)$, it's always sufficient to double the demand assignment $f(u, v)$ of u to v for each $v \in \mathrm{map}(u)$. If $\mathrm{map}(u)$ is only modified under the condition $0 < j_v < |N_{ud}[v]|$, (line 12 in Figure 4.4), then $\mathrm{map}(u)$ contains exactly the set of vertices that have partially served u. As mentioned above, since $\mathrm{rd}(u) < \frac{1}{2} \cdot d(u)$, it's sufficient to double the demand assignment in this case so $d(u)$ is completely satisfied. If $\mathrm{map}(u)$ is reassigned through the condition $j_v = 0$ for some stage, then we have $c(v) < \mathrm{rd}(u) \leq d(u)$. Since we assign this amount $c(v) \cdot \lfloor \mathrm{rd}(u)/c(v) \rfloor$ of residue demand of u to v, this leaves at most half of the original residue demand and u will be satisfied by doubling this assignment.

The second useful observation is that $n_j - n_{j+1} \geq \frac{1}{2}$, which means the decrease of n_j is at least a half. This will play a central role in bounding the overall cost.

Observation 2. We have $n_j - n_{j+1} \geq \frac{1}{2}$ for each $1 \leq j \leq m$.

Proof. For iteration j, $1 \leq j \leq m$, let u be the vertex of the maximum efficiency. Observe that $v_{u,1}$ will be satisfied after this iteration. By Observation 1, we have $\mathrm{rd}(v_{u,1})/d(v_{u,1}) > \frac{1}{2}$. The lemma follows. □

To see that the solution achieves the desired approximation guarantee, let's consider the cost taken by our algorithm which consists of the cost incurred by picking the vertex of the most efficiency replaced by the first case if necessary and the cost incurred by doubling the demand assignment in $\mathrm{map}(u)$ if needed. Let the former part be S_1 and the latter part be S_2. The important point to note is that S_2 is bounded above by S_1, for what we do is merely to satisfy the residue demand of a vertex, if there exists one, by doubling its previous demand assignment, which is incurred by some vertex of the most efficiency in some iteration.

Now we can start to develop a lemma similar to Lemma 3. For each iteration j, let u_j be the vertex of the maximum efficiency and OPT_j be the cost of the optimal solution for the remaining problem instance. Recall that $n_j = \sum_{u \in V} \mathrm{rd}(u)/d(u)$. Let $S_{1,j}$ be the cost incurred by the vertex of most efficiency in iteration j. Assume that the algorithm repeats for m iterations. We have the following lemma.

Lemma 6. For each j, $1 \leq j \leq m$, we have $S_{1,j} \leq \frac{n_j - n_{j+1}}{n_j} \cdot OPT_j$, where $n_j - n_{j+1}$ is the effectiveness covered by u_j in iteration j.

Proof. This proof is similar to the proof in Lemma 3. The optimality of our choice in each iteration is obvious since we assume that the elements of $N_{ud}[u]$ are sorted according to their original demands. Note that only in the case $c(u) < \mathrm{rd}(v_{u,1})$, the algorithm could possibly take more than one copy. In this case the efficiency of our choice remains unchanged since the cost and the effectiveness covered by u grows by the same factor. Therefore the efficiency of our choice, $(n_j - n_{j+1})/S_{1,j}$, is always no less than that of the optimal solution, which is n_j/OPT_j, and the lemma follows. $\square$

By Lemma 6 we have $\sum_{j=1}^{m} S_{1,j}$

$$\leq \sum_{j=1}^{m-1} \frac{n_j - n_{j+1}}{n_j} \cdot OPT_j + \frac{n_m}{n_m} \cdot OPT_m$$

$$\leq \left(\sum_{j=1}^{m-1} \frac{\lceil n_j - n_{j+1} \rceil}{\lfloor n_j \rfloor} + 1 \right) \cdot OPT,$$

since $\lfloor r \rfloor \leq r \leq \lceil r \rceil$ for any real number r and $OPT_j \leq OPT$ for each $1 \leq j \leq m$.

Lemma 7. $\sum_{j=1}^{m-1} \lceil n_j - n_{j+1} \rceil / \lfloor n_j \rfloor \leq 2 \ln n$

Proof. Note that by Observation 1 and Observation 2, we have $n_j > 1$ for all $j < m$. We will argue that this series together constitutes at most two harmonic series. By expanding the summand we have $\lceil n_j - n_{j+1} \rceil / \lfloor n_j \rfloor \leq$

$$\frac{1}{\lfloor n_j \rfloor} + \frac{1}{\lfloor n_j \rfloor - 1} + \dots + \frac{1}{\lfloor n_j \rfloor - \lceil n_j - n_{j+1} \rceil + 1}.$$

Since $\lfloor n_{j+1} \rfloor = \lfloor n_j - (n_j - n_{j+1}) \rfloor \leq \lfloor n_j \rfloor - \lfloor n_j - n_{j+1} \rfloor \leq \lfloor n_j \rfloor - \lceil n_j - n_{j+1} \rceil + 1$, the repetitions only occur at the first term and the last term if we expand the summation. By Observation 2, the decrease of n_j to n_{j+1} is at least half. Therefore, the term $\lfloor n_j \rfloor - \lceil n_j - n_{j+1} \rceil + 1$ will never occur more than twice in the expansion. We conclude that $\sum_{j=1}^{m-1} \lceil n_j - n_{j+1} \rceil / \lfloor n_j \rfloor \leq 2 \ln n$ $\square$

Since $S_1 \leq 2(\ln n + 1) \cdot OPT$ and S_2 is bounded above by S_1, we conclude the analysis in the following theorem.

Lemma 8. Algorithm *Split-Log-Approx* returns a $(4 \ln n + 2)$-approximation, where n is the number of vertices, for weighted capacitated domination problem with splittable demands.

Regarding time complexity it takes linear time to compute the efficiency of a vertex. Therefore it takes $O(n^2)$ time for each iteration. Since the number of iterations is bounded above by $2n$, the time complexity of the algorithm is $O(n^3)$. We can conclude this section by the following theorem.

Theorem 9. Algorithm *Split-Log-Approx* computes a $(4 \ln n + 2)$-approximation, where n is the number of vertices, for weighted capacitated domination problem with splittable demands in $O(n^3)$ time.

ALGORITHM *Split-Log-Approx*

```
1: $\mathrm{rd}(u) \longleftarrow d(u)$, and $\mathrm{map}(u) \longleftarrow \phi$ for each $u \in V$.
2: while there exist vertices with non-zero residue demand do
3:    // Searching for a vertex of the most efficiency
4:    Pick a vertex in $V$ with the most efficiency, say $u$.
5:    // Replace the demand assignment if in need
6:    if $j_u$ equals $0$ then
7:       Assign this amount $c(u) \cdot \left\lfloor \frac{\mathrm{rd}(v_{u,1})}{c(u)} \right\rfloor$ of residue demand of $v_{u,1}$ to $u$.
8:       $\mathrm{map}(v_{u,1}) \longleftarrow \{u\}$
9:    else
10:      Assign the residue demands of the vertices in $\{v_{u,1}, v_{u,2}, \ldots, v_{u,j_u}\}$ to $u$.
11:      if $j_u < |N_{ud}[u]|$ then
12:         Assign this amount $c(u) - \sum_{i=1}^{j_u} \mathrm{rd}(v_{u,i})$ of residue demand of $v_{u,j_u+1}$ to $u$.
13:         $\mathrm{map}(v_{u,j_u+1}) \longleftarrow \mathrm{map}(v_{u,j_u+1}) \cup \{u\}$
14:      end if
15:   end if
16:
17:   // doubling assignment if in need
18:   if there is a vertex $u$ with $0 < \mathrm{rd}(u) < \frac{1}{2} \cdot d(u)$ then
19:      Satisfy $u$ by doubling the demand assignment of $u$ to vertices in $\mathrm{map}(u)$.
20:   end if
21: end while
22: compute from the assignment the cost of the dominating set, and return the result.
```

Figure 4.4: The pseudo-code for the weighted splittable demand model.

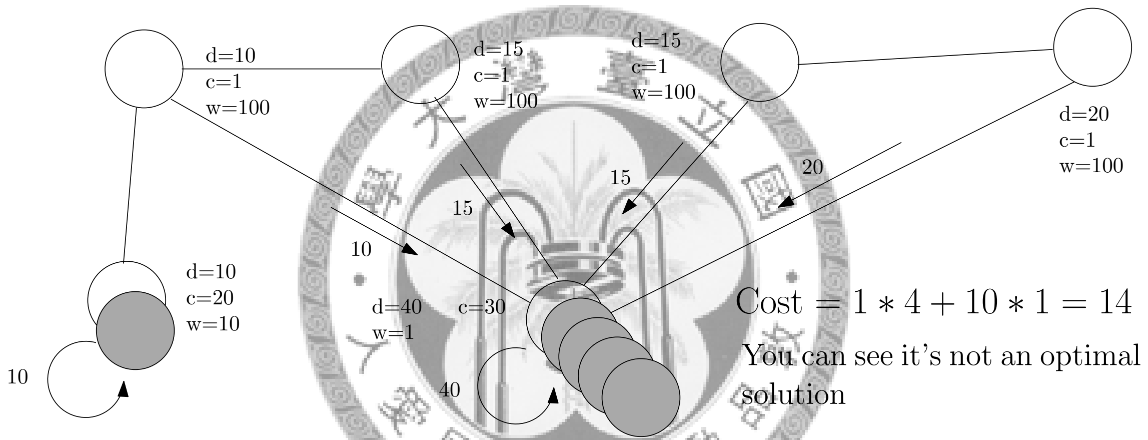


Figure 4.5: A feasible solution computed by our algorithm in splittable model.

Chapter 5

Unweighted Splittable Demand

In this chapter, we consider the *unweighted capacitated domination problem with splittable demand* and present a $(2\ln n + 1)$-approximation. In this special variation, the weight $w(v)$ of each vertex $v \in V$ is considered to be uniform and the cost of the capacitated domination multiset D corresponds to the total multiplicity of the vertices in D. First we perform a preprocessing to reduce the demand of each vertex. Second we slightly modify the algorithm to achieve the guarantee of $O(2\ln n + 1)$.

5.1 The Preprocessing on Problem Instance

For each $u \in V$, let g_u be the vertex in $N[u]$ with the maximum capacity. First, for each $u \in V$, we assign this amount $c(g_u) \cdot \left\lfloor \frac{d(u)}{c(g_u)} \right\rfloor$ of the demand of u to g_u. After that we update $d(u)$ as $\mathrm{rd}(u)$ to make a new instance.

Interestingly, although in this preprocessing we may reduce the demand greatly, the cost incurred from the preprocessing is rather cheap. Let the cost of the assignment incurred by the preprocessing be S, we have the following lemma.

Lemma 10. We have $S \le OPT$, where OPT is the cost of the optimal solution.

Proof. Notice that an optimal solution O^* for the relaxation of this problem, where fractional copies are allowed, can be obtained by assigning the demand $d(u)$ of u to g_u. Since $S \le O^*$ and $O^* \le OPT$, the lemma follows. $\square$

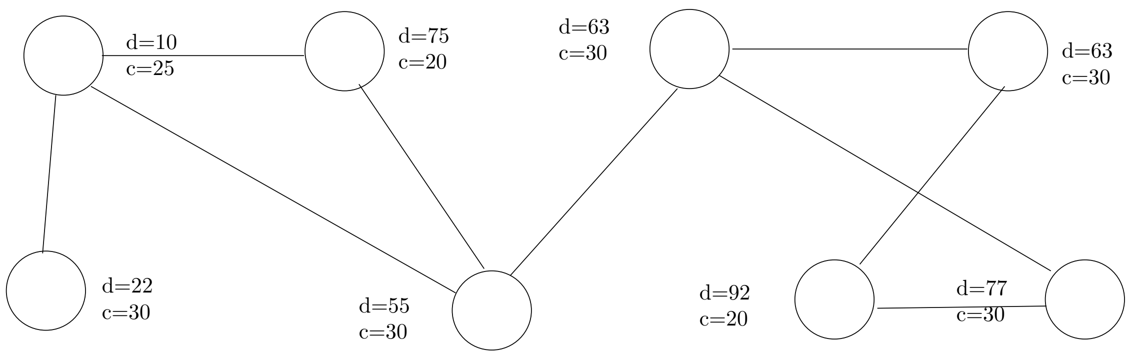


Figure 5.1: The graph before the greedy reduction.

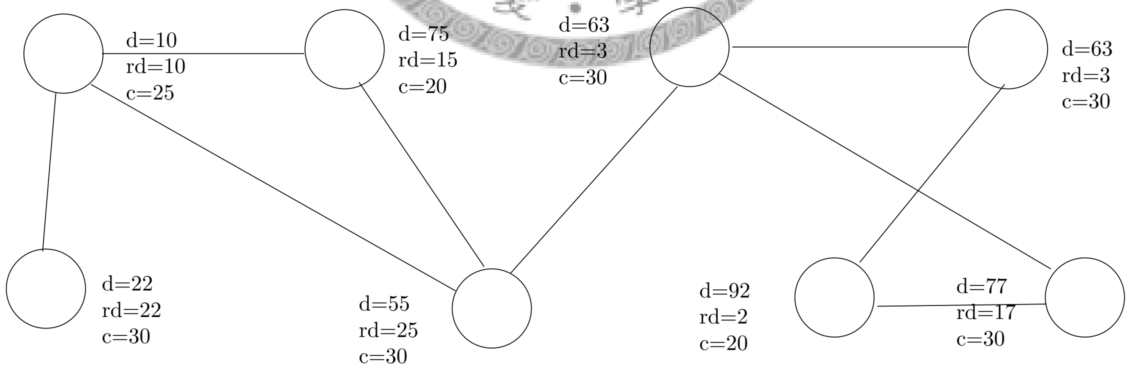


Figure 5.2: The graph after the greedy reduction.

```
ALGORITHM Unweighted-Split-Log-Approx
1: For each $u \in V$, assign $c(g_u) \cdot \left\lfloor \frac{d(u)}{c(g_u)} \right\rfloor$ demands of $u$ to $g_u$, where $g_u \in N[u]$ has the
   maximum capacity.
2: Reset the demands of the instance by setting $d(u) \longleftarrow \mathrm{rd}(u)$ for each $u \in V$.
3: while there exist vertices with non-zero residue demand do
4:     Pick a vertex in $V$ with the most efficiency, say $u$.
5:     Assign the residue demands of the vertices in $\{v_{u,1}, v_{u,2}, \ldots, v_{u,j_u}\}$ to $u$.
6:     if $j_u < |N_{ud}[u]|$ then
7:         Assign this amount $c(u) - \sum_{i=1}^{j_u} \mathrm{rd}(v_{u,i})$ of the residue demand of $v_{u,j_u+1}$ to $u$.
8:     end if
9:
10:    if there is a vertex $u$ with $0 < \mathrm{rd}(u) < d(u)$ then
11:        Satisfy $u$ by assigning the residue demand of u to $g_u$.
12:    end if
13: end while
14: compute from the assignment the cost of the dominating set, and return the result.
```

Figure 5.3: The pseudo-code for the unweighted splittable demand model.

Note that after the preprocessing, we have $d(u) \leq c(g_u)$ for each $u \in V$, which we will assume to be true in the rest of this Chapter.

Figure 5.1 and Figure 5.2 demonstrate examples of a problem instance before and after the preprocessing.

5.2 The Algorithm for Capacitated Domination Problem with Unweighted Splittable Model

The algorithm presented in Chapter 4 is slightly modified. In particular, when handling the second special case: Whenever $\mathrm{rd}(u) < d(u)$ for some vertex $u \in V$, we immediately assign the residue demand of u to g_u.

A high-level description of this algorithm is presented in Figure 5.3.

5.3 Analysis

The analysis is very similar to what we have in Chapter 4. As a result of the preprocessing we have the following property.

Observation 3. We have $n_j - n_{j+1} \geq 1$ for each $1 \leq j \leq m$.

Proof. Observe that in each iteration, at least one vertex is satisfied and the residue demand of each unsatisfied vertex is equal to its original demand. □

Clearly, S_2 is bounded above by S_1, as we always take one copy for the vertex of the most efficiency and at most one copy to satisfy residue demand of a vertex in each iteration. By Observation 3 and the fact that n_j is integral for each $1 \leq j \leq m$, we have

$$\sum_{j=1}^{m} S_{1,j} \leq \sum_{j=1}^{m} \frac{n_j - n_{j+1}}{n_j} \cdot OPT_j \leq \ln n \cdot OPT,$$

and $S + \sum_{j=1}^{m} (S_{1,j} + S_{2,j}) \leq (2 \ln n + 1) \cdot OPT$.

We conclude the result of this chapter as the following theorem.

Theorem 11. Algorithm *Unweight-Split-Log-Approx* computes a $(2\ln n{+}1)$-approximation for weighted capacitated domination problem with unsplittable demands in $O(n^3)$ time, where n is the number of vertices.

Chapter 6

Concluding Remarks

In this thesis, we consider the *capacitated domination* problem, which is a generalization of the well-known *domination* problem and which models a service-requirement assignment scenario. In terms of polynomial time approximations, we have presented logarithmic approximation algorithms with respect to different demand assignment models for this problem on general graphs. In traditional domination problem there are two kinds of approximation guarantee, $\ln n$ and Δ, where Δ is degree of the input graph. Prior to this work, Kao et al. provided a $\Delta + 1$-approximation [16]. The table below shows that together with previously known results on generally approximating this problem, our work establishes the corresponding approximating results similar to the well-known approximations of the traditional *domination* problem and closes the problem of generally approximating the optimal solution.

Tabular of Approximation Result

Traditional Domination Problem	Capacitated Domination Problem
$\ln n$-approximation by Johnson et al.[7, 15, 19]	$O(\ln n)$-approximation by this work.
Δ-approximation by D.S. Hochbaum[14]	$\Delta + 1$-approximation by Kao et al.[16]

Chapter 7

Appendix

An Implementations of Capacitated Domination Problem with GUI for demonstrations can be found in the following URL.

http://homepage.ntu.edu.tw/ r97922104/CDP.rar

This problem is written in Java language. One may need Java Runtime Environment for execution. The Java Runtime Environment can be found in the following URL.

http://www.java.com/en/download/

A snapshot of the problem is shown in Figure 7.1.

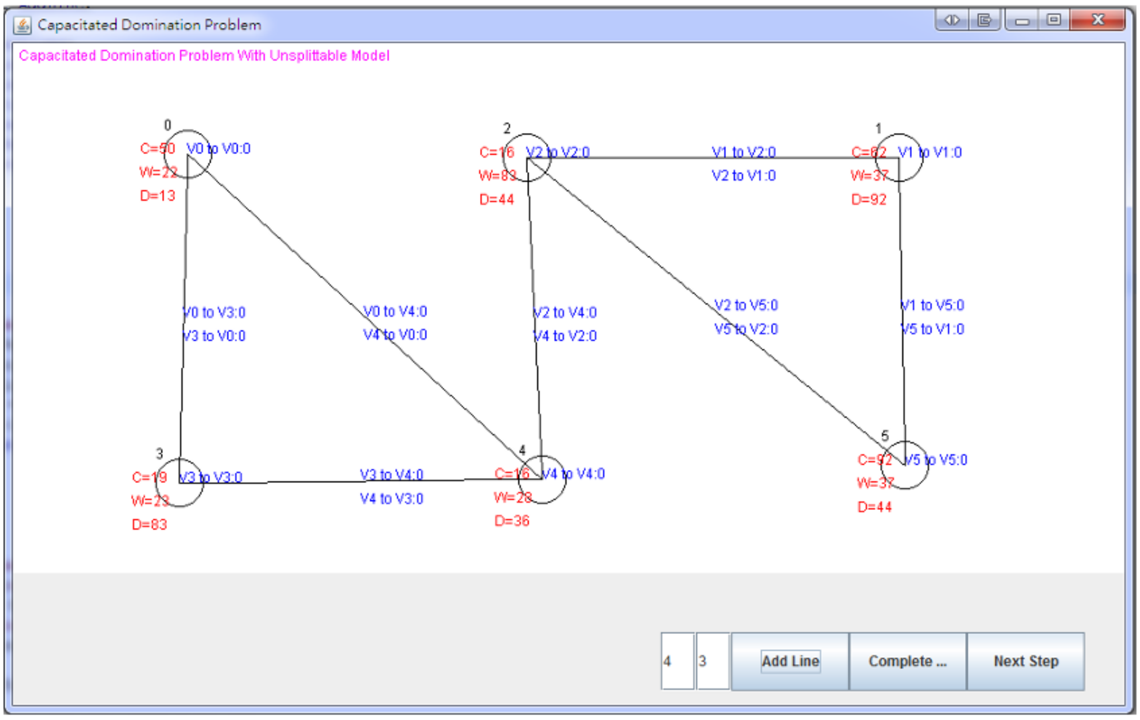


Figure 7.1: A snapshot of the problem.

Bibliography

[1] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002.

[2] Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, 2004.

[3] Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.

[4] Judit Bar-Ilan, Guy Kortsarz, and David Peleg. Generalized submodular cover problems and applications. *Theor. Comput. Sci.*, 250(1-2):179–200, 2001.

[5] H. L. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3), 2008.

[6] Julia Chuzhoy. Covering problems with hard capacities. *SIAM J. Comput.*, 36(2):498–515, 2006.

[7] Václav Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[8] Michael Dom, Daniel Lokshtanov, Saket Saurabh, and Yngve Villanger. *Capacitated Domination and Covering: A Parameterized Perspective*, volume 5018 of *Lecture Notes in Computer Science*, pages 78–90. Springer Berlin/Heidelberg, 2008.

[9] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[10] Fedor V. Fomin and Dimitrios M. Thilikos. Dominating sets in planar graphs: Branch-width and exponential speed-up. *SIAM J. Comput.*, 36(2):281–309, 2006.

[11] Sudipto Guha, Refael Hassin, Samir Khuller, and Einat Or. Capacitated vertex covering. *J. Algorithms*, 48(1):257–270, 2003.

[12] Teresa W. Haynes, Sandra M. Hedetniemi, Stephen T. Hedetniemi, and Michael A. Henning. Domination in graphs applied to electric power networks. *SIAM J. Discret. Math.*, 15(4):519–529, 2002.

[13] Teresa W. Haynes, Stephen Hedetniemi, and Peter Slater. *Fundamentals of Domination in Graphs (Pure and Applied Mathematics)*. Marcel Dekker, 1998.

[14] D.S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982.

[15] David S. Johnson. Approximation algorithms for combinatorial problems. In *STOC '73: Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, pages 38–49, New York, NY, USA, 1973. ACM.

[16] Mong-Jen Kao, Chung-Shou Liao, and D. T. Lee. Capacitated domination problem. *Algorithmica*, 2009.

[17] Ton Kloks. *Treewidth. Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer Berlin/Heidelberg, 1994.

[18] Chung-Shou Liao and Der-Tsai Lee. *Power Domination Problem in Graphs*, volume 3595 of *Lecture Notes in Computer Science*, pages 818–828. Springer Berlin / Heidelberg, 2005.

[19] Lászlo Lovász. On the ratio of optimal integral and fractional covers, 1975.

[20] Fred S. Roberts. *Graph Theory and Its Applications to Problems of Society*. 1978.

[21] P.-J. Wan, K. M. Alzoubi, and O. Frieder. A simple heuristic for minimum connected dominating set in graphs. *International Journal of Foundations of Computer Science*, 14(2):323–333, 2003.