NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

UNDERGRADUATE THESIS

The Maximum Rooted Connected Expansion problem

Nikolaos P. Theodorou

Supervisor : **Vassilis Zissimopoulos,** Professor NKUA

ATHENS

SEPTEMBER 2019

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Το πρόβλημα της μέγιστης ριζικής συνεκτικής επέκτασης.

Νικόλαος Π. Θεοδώρου

Επιβλέπων : **Βασίλης Ζησιμόπουλος,** Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2019

UNDERGRADUATE THESIS

Maximum Rooted Connected Expansion

Nikolaos P. Theodorou
S.N.: 1115201000030

SUPERVISOR: **Vassilis Zissimopoulos ,** Professor NKUA

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μια έρευνα πάνω στο ελάχιστο συνδεδεμένο σύνολο κυριαρχίας, στα δέντρα επικάλυψης με μέγιστο πλήθος φύλλων και το πρόβλημα της μέγιστης ριζικής συνεκτικής επέκτασης.

Νικόλαος Π. Θεοδώρου

Α.Μ.: 11152001000030

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Βασίλης Ζησιμόπουλος ,** Καθηγητής ΕΚΠΑ

ABSTRACT

Graph theory has many applications in modern world. Our study on it crossed paths with an intresting problem called Prefetching. Prefetching constitutes a valuable tool toward the goal of effiecient Web Surfing. An important issue in prefetching is the tradeoff between the amount of network's resources wasted by the prefetching and the gain of time. For instance, in the Web, browsers may download documents in advance while a Web surfer is surfing on the Web. Since the Web surfer follows the hyperlinks in an unpredictable way, the choice of the Web pages to be prefetched must be computed online. The question is then to determine the minimum amount of resources used by prefetching that ensures that all documents accessed by the Web surfer have previously been loaded in the cache. In this regard, prefetching can be modeled as a two-player combinatorial game.

Motivated by the above Sigalas et al. considered the following maximization problem to which they refer to as the Maximum Rooted Connected Expansion (MRCE) problem which is NP-hard. Given a graph G and a root node u_0 , we wish to find a subset of vertices S such that S is connected, S contains u_0 and the ratio $N[S]/|S|$ is maximized, where $N[S]$ denotes the closed neighbourhood of S , that is $N[S]$ contains all nodes in S and all nodes with at least one neighbour in S .

We further discuss their approach on the way of solving this problem through an approximation algorithm. We studied other significant problems on graph theory like Connected Dominating Set, Maximum Leaf Spanning Tree and their special case problems so as to understand the structure and connection between all those and MRCE. As result of that study we mapped those problems according to their connection and interaction between them. Our contribution is a greedy algorithm for the MRCE which we believe through our experimental analysis could eventually lead to a small approximation, result.

SUBJECT AREA: Algorithms, Graph Theory, Combinatorial Optimization

KEYWORDS: prefetching, connected dominating set, expansion, ratio, maximum leaf

ΠΕΡΙΛΗΨΗ

Η θεωρία γραφημάτων έχει πολλές εφαρμογές στον σύγχρονο κόσμο. Η μελέτη μας σε αυτήν συνέπεσε με ένα ενδιαφέρον πρόβλημα που ονομάζεται προανάκληση. Η προανάκληση αποτελεί ένα πολύτιμο εργαλείο για το στόχο της αποτελεσματικής πλοήγησης στο διαδύκτιο. Ένα σημαντικό ζήτημα στην προανάκληση είναι η ανταλλαγή μεταξύ του ποσού των πόρων του δικτύου που χάνονται από την προανάκληση και το κέρδος του χρόνου. Για παράδειγμα, στον ιστό, τα προγράμματα περιήγησης ενδέχεται να κατεβάζουν εκ των προτέρων έγγραφα, ενώ ένας διαδικτυακός χρήστης πλοηγείται στο διαδίκτυο. Δεδομένου ότι ο διαδικτυακός χρήστης ακολουθεί τους υπερσυνδέσμους με έναν απρόβλεπτο τρόπο, η επιλογή των ιστοσελίδων που πρέπει να προανακλήθουν πρέπει να υπολογιστεί ηλεκτρονικά. Το ερώτημα είναι τότε να προσδιοριστεί το ελάχιστο ποσό των πόρων που χρησιμοποιούνται από την προανάκληση, που εξασφαλίζει ότι όλα τα έγγραφα που έχει πρόσβαση ο διαδικτυακός χρήστης, έχουν προηγουμένως φορτωθεί στην κρυφή μνήμη. Από την άποψη αυτή, η προανάκληση μπορεί να διαμορφωθεί ως συνδυαστικό παιχνίδι δύο παικτών.

Με γνώμονα τα παραπάνω οι Λάμπρου και συνεργάτες θεώρησαν το ακόλουθο πρόβλημα μεγιστοποίησης στο οποίο αναφέρονται ως πρόβλημα Maximum Rooted Connected Expansion (πρόβλημα της μέγιστης ριζικής συνεκτικής επέκτασης), MRCE που είναι NP-Hard. Με βάση ένα γράφημα G και έναν κόμβο ρίζας v_0, θέλουμε να βρούμε ένα υποσύνολο κορυφών S που να ειναι συνδεδεμένο, να περιέχει το v_0 και να μεγιστοποιείται ο λόγος $N[S]/|S|$, όπου το $N[S]$ δηλώνει την κλειστή γειτονιά του S, δηλαδή περιέχει όλους τους κόμβους του S και όλους τους κόμβους του εκτός του S με τουλάχιστον έναν γείτονα εντός του S.

Αναλύουμε περαιτέρω την προσέγγισή τους σχετικά με τον τρόπο επίλυσης αυτού του προβλήματος μέσω ενός προσεγγιστικού αλγόριθμου. Μελετήσαμε άλλα σημαντικά προβλήματα στη θεωρία των γραφημάτων και τα ειδικά προβλήματά τους για να κατανοήσουμε τη δομή και τη σχέση μεταξύ όλων αυτών και του MRCE. Ως αποτέλεσμα αυτής της μελέτης χαρτογραφήσαμε αυτά τα προβλήματα σύμφωνα με τη σύνδεσή τους και την αλληλεπίδραση μεταξύ τους. Η συνεισφορά μας είναι ένας άπληστος αλγόριθμος για το MRCE που πιστεύουμε μέσω της πειραματικής μας ανάλυσης θα μπορούσε ενδεχομένως να οδηγήσει σε ένα καλό προσεγγιστικό αποτέλεσμα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Αλγόριθμοι, Θεωρία Γραφημάτων, Συνδυαστική Βελτιστοποίηση

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Προανάκληση, μεγιστη ριζική συνεκτική επέκταση, δέντρο μέγιστου βαθμού φύλλων

ΕΥΧΑΡΙΣΤΙΕΣ (ή AKNOWLEDGMENTS)

Για την εκπόνηση της παρούσας Πτυχιακής Εργασίας για την οποία εργάστηκα με κάποιες μεγάλες παύσεις τα τελευταία 2 χρόνια, θα ηθέλα να ευχαριστήσω την οικογένεια μου, την κόπελα μου και τους φίλους μου που μου δώσαν κουράγιο και ηρεμία κάνοντας υπομονή όλη αυτήν την περίοδο. Επίσης θα ήθελα να ευχαριστήσω τον διδάκτωρα Ιωάννη Σιγάλα για την χρήσιμη συνεισφορά του και την άμεση ανταπόκριση του ανεξαρτήτως των πιο σημαντικών του υποχρεώσεων του. Τέλος θα ήθελα να ευχαριστήστω θερμά τον επιβλέποντα καθηγητή μου κ.Ζησιμόπουλο. Ο κ.Ζησιμόπουλος αρχικά με εμπιστεύτηκε και μου έδωσε τα κατάλληλα ερεθίσματα και κίνητρα για να ασχοληθώ με πάθος και αγάπη για αυτήν την εργασία ανεξάρτητα απο το αν οι ιδέες μας ήταν επιτυχημένες ή όχι. Με το πέρας αυτής της εργασίας πήρα πόλλα μαθήματα αλλά το πιο σημαντικό ήταν ότι μια ιδέα – προσπάθεια πρέπει πάντα να την ολοκληρώνουμε με σωστό και δομημένο τρόπο, καθώς μέσα απο αυτήν την διαδικασία θα βγαίνουμε πάντα κερδισμένοι ανεξαρτήτως του αποτελέσματος.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

1. INTRODUCTION

1.1 Introduction

Graph Theory is a very important discipline which lies between Computer Science and Mathematics, and involves the study of graphs, which are mathematical structures used to model pairwise relations between objects. One of the most important uses of Graph Theory is the capability to model and give a unified formalism for many very different looking problems coming from everyday life and a wide variety of sciences. Graph Theory has many applications, the one with most importance for us is networking. Networking has real life applications using Graphs i.e. connecting with friends on social media, using GPS/Google maps, surfing through the internet. In our thesis we studied the last case in its optimized version called Prefetching [1].

Prefetching is a basic technique in computer science. It exploits the parallelism between the execution of one task and the transfer of information necessary to the next task, in order to reduce waiting times. The classical instance of the problem occurs in CPU, where instructions and data are prefetched from the memory while previous instructions are executed. The modern instance occurs in the Web, where browsers may download documents connected to the currently viewed document (Web page, video,etc.) while it is being read or viewed. Accessing the next document appears to be instantaneous to the user, and gives the impression of a large navigation speed. For this reason, link prefetching has been proposed as a draft Internet standard by Mozilla. However, prefetching all documents that can be accessed in the current state may exceed networking capacities, or at least, result in a waste of bandwidth since most of the alternatives will not be used. Hence, it is necessary to balance the gain of time against the waste of networking resources. Local storage memory is also a potential issue, and prefetching is classically associated with the question of cache management. However, memory in modern computers is not scarce anymore, which makes network resources the critical ones. The models developed so far in the literature to study prefetching problems are based on the execution digraph where the nodes represent the tasks (e.g., Web pages) and arcs model the fact that a task can be executed once another has been done (e.g., arcs represent hyperlinks that can be followed from a Web page). The execution of the program or the surfing of the Web then corresponds to a path in the execution digraph. The quantitative optimization of prefetching will then be based on some cost function defined on paths, reflecting for instance the inconvenience of waiting

for some information while executing the tasks or surfing the Web, and possibly taking into account the consumption of network or memory resources. The related dimensioning problem consists in determining how much network bandwidth should be available so that the prefetching performance stays within some predetermined range.

In chapter 2 we present our study and thoughts on important problems in Graph Theory as well as giving an idea on how we can utilize the combinatorial optimization property of some problems. In Chapter 3 we formally present the Maximum Rooted Connected Expansion problem. We first present Lamprou et al. approach [2], solution and results. We then explain our thought process and demonstrate our algorithm and results. Finally we conclude with some ideas and future work directions.

1.2 Preliminaries

In the following two subsections we define and explain some basic notions that are used in this thesis concerning Graph Theory as well as the Surveillance Game. It is aimed for helping readers who are not familiar with these notions to have a better understanding of the text and how the Maximum Rooted Connected Expansion Problem occurred. In every section we will introduce the preliminaries necessary for each problem.

1.2.1 Graph Theory

An (undirected) graph G is a defined as a pair of two sets: the set of vertices V and the set of edges E, denoted $G=(V,E)$. An edge is an unordered pair of two vertices. Therefore, the set of edges is a subset of the of all possible unordered pairs of vertices $\{(u,v):u,v\in V\}$ and we write $e=(u,v)$ when edge e connects vertices u and v of V. Unless otherwise stated, all graphs are simple, which means that they contain neither loops nor parallel edges.

A path in a graph G is a sequence of vertices $u_{0,}u_{1,}\ldots,u_k$ where $u_i\neq u_j\forall\, i,j\in[0,k]$. On the contrary If $u_0=u_k$ then the sequence of vertices is called a *cycle*.

A graph G is a spanning tree T_s if between any two vertices $u_i,u_j\in V$ there exists exactly one path. In other words, tree-graph is any connected graph, without *cycles*.

The degree of a vertex G of a graph, denoted $d(u)$, is the number of edges incident to the vertex.

A graph G is connected if each pair of vertices $u_i, u_j \in V$ is joined by a *path*. On the contrary if there exist vertices which cannot be joined by a path the graph is disjoint.

1.2.2 Surveillance Game

The surveillance problem deals with the following two players game in an n-node graph $G=(V,E)$ with a given starting vertex $v_0 \in V$. There are two players, fugitive and observer. The fugitive wants to escape the control of an observer whose purpose is to keep the fugitive under constant surveillance. Let $k \geq 1$ be a fixed integer. The game starts when the fugitive stands at v_0 which is initially marked. Then, turn by turn, the observer controls, or marks, at most k vertices and then the fugitive either moves along an edge to a (out-)neighbor of her current position, or skip her move. In other words, at every step of the game the observer enlarges observable part of the graph by adding to it k, not necessarily adjacent, vertices. His task is to ensure that the fugitive is always in the observable area. Note that, once a vertex has been marked, it remains marked until the end of the game. The fugitive wins if, at some step, she reaches an unmarked vertex and the observer wins otherwise. That is, the game ends when either the fugitive enters an unmarked vertex (and then she wins) or all vertices have been marked (and then observer wins). More formally, a k-strategy (for the observer) is a function σ that assigns a subset $S \subseteq V, |S| \leq k$, to any configuration (M,f) of the game where $M \subseteq V$ is the set of the vertices that have already marked before this step of the game, $f \in M$ is the current position of the fugitive, and $S=\sigma(M,f)$ is the set of vertices to be marked at this step. Clearly, we can restrict our investigation to the case where $\sigma(M,f) \subset V \setminus M$ and $|\sigma(M,f)|=k$ or $\sigma(M,f)=V \setminus M$. That is, at each step, the observer has interest to mark as many unmarked vertices as possible. In particular, a game consists of at most n/k steps. A k-strategy is winning if it allows the observer to win whatever be the walk followed by the fugitive. Note that any winning strategy must ensure that $N(f) \setminus M \subseteq \sigma(M,f)$ for any $M \subseteq V$, $f \in M$. The surveillance number of G, denoted by $sn(G,v_0)$, is the least k such that there is a winning k-strategy in G starting from v_0.

Let $N[S]$ stand for the closed neighborhood of S, i.e., $N[S]$ includes all nodes in S and all nodes with at least one neighbor in S. Fomin et al. prove in [1] (Theorem 20) that, for any graph G and root v_0, it holds $sn(G,v_0) \geq max \frac{|N[S]|-1}{|S|}$, where

the maximum is taken over all subsets S that induce a connected subgraph of G containing v_0. Moreover, equality holds in case G is a tree. That is, a ratio of the form $\frac{|N[S]|}{|S|}$ (minus one removed for clarity) provides a good lower bound and possibly in many occasions a good prediction on the prefetching load necessary to satisfy an impatient Web surfer. Hence, in this paper, we believe it is worth to independently study the problem of determining $max\frac{|N[S]|}{|S|}$ where the maximum is taken over all subsets S inducing a connected subgraph of G containing v_0. We refer to this problem as the Maximum Rooted Connected Expansion problem since we seek to find a connected set S (containing the root v_0) maximizing its expansion ratio in the form of $\frac{|N[S]|}{|S|}$.

2. STUDY ON GRAPH THEORY

We now proceed to formally define the problems we consider in this paper.

2.1 Connected Dominating Set

We will now formally adress the Connected Dominating Set problem and present some algorithms for its approximate solution.

2.1.1 Problem Statement

A connected dominating set (CDS) in a graph is a subset of vertices that induces a connected subgraph, and is also a dominating set at the same time. A dominating set is a subset of vertices such that every node in the graph, is either in the dominating set, or adjacent to a node in the dominating set. Finding a minimum connected dominating set is NP-hard as described by Guha and Khuller [3] .We state the problem as follows :

Problem Statement 2 (Connected Dominating Set, CDS)

Input : A graph $G=(V,E)$.

Output : A dominating set S where all vertices of G are either in S or dominated (connected) from one vertex in S.

2.1.2 Algorithms

We will shortly present some algorithms for the problem as described by Khuller and Yang in [4]. We will later discuss their importance.

Global Algorithm for CDS

The global algorithm runs in two phases. Initially, all nodes are colored white. In the first phase, the algorithm iteratively adds a node to the solution, colors it black and all its adjacent white nodes gray. A piece is defined as a white node or a black connected component. A new node is chosen to be colored black to get maximum reduction in the number of pieces. This phase ends when no such node exists that can give non-zero reductions. At this time, there are no white nodes left. Intuitively speaking, black nodes are selected nodes, gray nodes are nodes that are dominated, i.e. adjacent to black nodes. In the second phase, we start with a dominating set that consists of several black components that we need to connect. The connection is done by recursively connecting pairs of black components with a chain of vertices, until there is only one black component, which will be our final solution.

Preliminaries for 1-hop Local Information Algorithms

Instead of using information of the entire graph, it only relies on information within 1-hop to the nodes chosen in the solution. The formal definition of local information is as follows.

In undirected graph, we denote the distance between u and v in a graph as $d(u,v)$. It is the length of shortest path from u to v. $d(u,S)$ is defined to be $min_{v \in S} d(u,v)$.

We now define the local neighborhood of some node, or a set of nodes. Given a set of nodes S in graph G, the $r-hop$ neighborhood around S is the induced subgraph of G containing all nodes v such that $d(v,S) \leq r$. We denote the $r-hop$ neighborhood as $N^r(S)$. When there is no confusion, we use the same notation to denote the set $N^r(S) = \{ v \mid d(v,S) \leq r \}$. We also denote the dominated set of a node v as the degree of node v. We can find out that $N^1(S)$ is the degree of S. We can see that $N^r(S)$ is the dominated set of $N^{r-1}(S)$.

An algorithm with local information uses information only within the local neighborhood of the nodes it has chosen. To be specific if the set of nodes that an algorithm has chosen is S and we have r-hop local information, then we know the induced graph of $N^r(S)$. From an arbitrary starting node, nodes are added iteratively. For each loop in this algorithm, one chooses a node, or a node and one of its neighbors. This means we need knowledge of 1-hop neighborhood to maximize the number of newly covered nodes, which explains why it uses 1-hop of local information.

1-hop Local Information Algorithm for CDS

This algorithm chooses only one gray node to maximize the number of newly covered nodes, and in addition selects one of the newly covered nodes uniformly at random. The maximization process only requires 1-hop neighborhood information. **Algorithm 1.**

Improved 1-Hop Local Information Algorithm

Instead of using the largest degree in the graph, every time when calculating propability p, we use the largest degree in the explored graph. In another world, the largest degree in $N^1(S)$. **Algorithm 2**

Algorithm 1: *Randomized CDS with 1-hop Local Information*

Input: Graph $G=(V,E)$.

Output: A connected dominating set S.

1: $s \leftarrow$ an arbitrary node

2: $S \leftarrow \{s\}$

3: **while** S *is not a dominating set* **do**

4: $v \leftarrow$ a node in $N^1(S)$ that maximizes the number of newly dominated nodes;

5: $u \leftarrow$ a uniformly randomly chosen node from $N^1(v) - N^1(S)$, i.e. the newly dominated nodes

6: $S \leftarrow S \cup \{(u,v)\}$

Algorithm 2: *Improved Algorithm for CDS with 1-hop Local Information*

Input: Graph $G=(V,E)$.

Output: A connected dominating set S.

1: $s \leftarrow$ an arbitrary node

2: $S \leftarrow \{s\}$

3: **while** S *is not a dominating set* **do**

4: $d \leftarrow$ the largest degree in $N^1(S)$

5: $p \leftarrow \frac{1}{\sqrt{H(d)}}$

6: $v \leftarrow$ a node in $N^1(S)$ that maximizes the number of newly covered nodes;

7: $S \leftarrow S \cup \{u\}$

8: **if** *with probability* p **then**

9: $u \leftarrow$ a node from the newly covered nodes uniformly at random

10: $S \leftarrow S \cup \{u\}$

2.1.3 Results and Discussion

For the first algorithm , Global Algorithm for CDS the approximation ratio for this algorithm is $H(\Delta)+2$, where $H(n)$ is harmonic function. The refined version of this greedy algorithms pseudo-coloring is called labeling. Labeling is used widely on greedy approximation algorithms and also on the MRCE greedy algorithm. On the second algorithm 1-hop Local Information Algoritm we get an approximation ratio of $2(H(\Delta)+1)$. This algorithm, as we will discuss seems very familiar with the 1-LOT

algorithm used in Maximum Leaf Spanning Tree problem. We now understand the close connection between those two problems and that sometimes we can approach both of them with the same techniques. Last is the improved 1-hop algorithm which gives a $H(\Delta)+2\sqrt{H(\Delta)}+1$ approximation. One thing we can take from their contribution is that if we find ways of slightly adjusting our algorithms by making small corrections, then the difference in approximation ratios can be significant.

2.2 Maximum Leaf Spanning Tree

We will now formally adress the Maximum Leaf Spanning Tree problem and present two algorithms for its approximate solution.

2.2.1 Problem Statement

Given a simple, undirected graph $G=(V,E)$, suppose we wish to find a spanning tree of G with the maximum number of leaves. The maximum-leaf spanning tree problem is proven to be NP-complete [5]. This suggests that no polynomial time algorithm is likely. So we seek fast approximation algorithms that provide a good worst case performance ratio on the quality of the solution. In our thesis, we address two approximation algorithms for this problem in order to understand their importance. The first algorithm uses the simple technique of local optimization: perform a series of local-improvement steps until a local optimum is reached. The other one uses a greedy approach that is based on the domination degree of nodes in G, combined with some required properties that induced sub-trees created from those nodes must meet. We state the problem as follows :

Problem Statement 2 (Maximum Leaf Spanning Tree, MLST)

Input : An arbitrary spanning tree $T_s=(V,E_s)$ from a graph $G=(V,E)$.

Output : A maximum leaf spanning tree $T_{OPT}=(V,E_{OPT})$.

We will study the first algorithm by the name of **1-LOT** and the second a greedy algorithm the **Leafy Forest**. Both of these algorithms have been applied to provide heuristic solution for a variety of hard problems in combinatorial optimization. Thus on later chapters we will make a small discussion on combining them with other algorithms for better results.

2.2.2 Local Optimum Tree

Following a small presentation of the first algorithm the thought process and its results.

Problem Statement / Idea

The idea of this algorithm as descibed by Lu and Ravi in [6] is performing local changes that increase the number of leaves in the resulting spanning tree. For the problem at hand, there is a natural notion of such local change ,namely, a **swap.** Thus swapping a tree edge for a non-tree edge. This notion can also be extended to include swapping many tree edges for an equal number of non-tree edges in a single local-change operation.The algorithm performs such local changes that increase the number of leaves until no improvement in the solution is possible and output a locally optimal solution as the approximate solution. By varying the limit on the number of tree edges that can participate in a single local-change operation, we derive a series of approximation algorithms. Increasing this limit corresponds to allowing for more powerful local-improvement steps. We will address the 1-change algorithm and its results.

Preliminaries

We use the term degree of a node to refer to its degree in the tree under consideration. So leaves are just nodes of degree one. A node is internal if it is not a leaf. We call a node of degree greater than two a high degree node. A leaf is *specia*l if it is adjacent to a high degree node. All other leaves are termed *normal*. Thus *normal* leaves are exactly the leaves that are adjacent to nodes of degree two in the tree. A tree path containing only nodes of degree two is called a 2-path. Its length is the number of nodes in it. A 2-path is $short$ if its length is one; otherwise it is $long$. We shall refer to an edge $e=(u,v)$ as an edge between u and v.

For a spanning tree T, we use $L(T)$ to denote the number of leaves in T. We omit the T where it is understood. We use n_i to denote the number of nodes of degree i. Suppose the input graph G has n nodes. We define N_i, the number of nodes with degree at least i. Thus for instance, N_3 is the number of high degree nodes. The number of $long$ and $short$ 2-paths are represented by P_l and P_s respectively. Let s and n be the number of *special* and *normal* leaves respectively.

Algorithm

Our algorithm starts with an arbitrary spanning tree T of the given graph G. Let $e=(u,v)$ be an edge in $G-T$ and f be an edge in the unique tree path of T connecting u and v. We call making e a tree edge and f a non-tree edge an (edge)

change $(e;f)$ with respect to T. The spanning tree obtained by applying change $(e;f)$ on T is denoted by $T(e;f)$. If $T(e;f)$ has more leaves than T ,then we call the change $(e;f)$ an improvement. In this case we call that a 1-improvement due to the fact that we are ***swapping*** one edge a time (1-change). The algorithm terminates when spanning tree T does not admit a 1-improvement. Despite that the same tree could admit a 2-improvement but we wont address this case of k -changes in this thesis.

In Figure (1) we see an example of local improvement. In a graph G (a), we take an arbitrary spanning tree T (b). The **swapping**, change of $e(v_0,v_1)$ with $e(v_0,v_2)$ is a 1-improvement on T resulting in T_1 (c) with one more leaf as shown in (b).

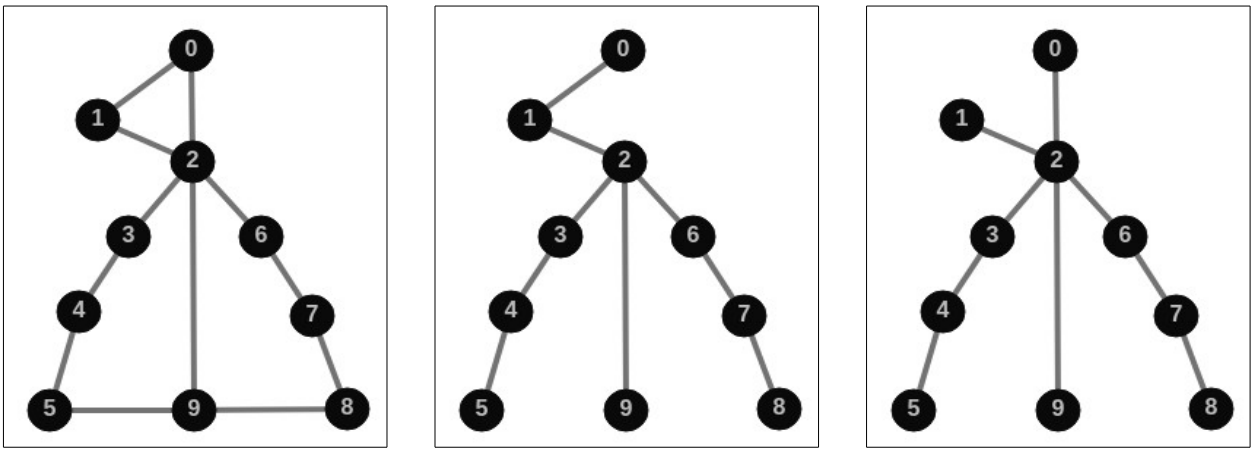


Figure (1): Example of local improvement

Properties

By definition, a 1-LOT has the following properties :

P1 In a 1-LOT T , any cycle formed by adding a non-tree edge between a leaf and an internal node does not contain two adjacent nodes of degree two.

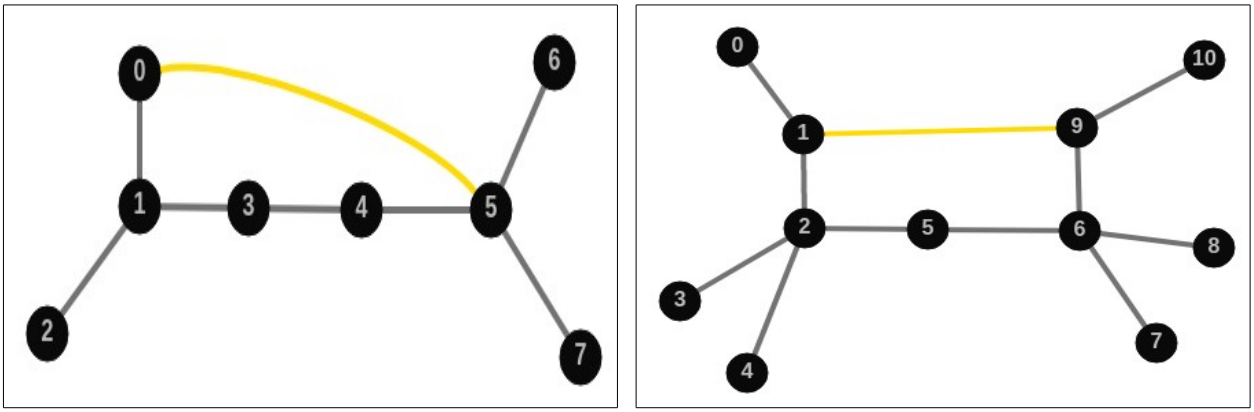
P2 In a 1-LOT T , any cycle formed by adding a non-tree edge between two internal nodes does not contain a node of degree two.

P1 and **P2** together are necessary and sufficient conditions for 1-optimality.

We prove the contrapositive and show that if a spanning tree is not a 1-LOT then it violates either **P1** or **P2**. If a spanning tree is not a 1-LOT then there exists a 1-improvement for this spanning tree. It is easy to see that any non-tree edge incident on two leaves cannot be involved in any 1-improvement. So any non-tree edge involved in

a 1-improvement must be an edge between an internal node and a leaf or an edge between two internal nodes. In the first case, the tree violates property **P1** and in the second case, it violates **P2**.

In Figure(2) we illustrate the properties P1 and P2 for 1-LOTs. In the figure below, dark edges represent tree edges and the yellow edge is a non-tree edge violating the property P1 or P2. In both cases, a 1-improvement by **swapping** $e(v_3, v_4)$ with $e(v_0, v_5)$ can be identified in P1 and $e(v_5, v_6)$ with $e(v_1, v_9)$ in P2.



(P1) **(P2)**

Figure (2): Properties of 1-LOT

Results

The main results of their analysis is that local-improvement algorithms that use 1-changes and 2-changes have performance ratios of 5 and 3 respectively. Althougth we didnt address the 2-changes algorithm neither the k -changes which its time complexity is intolerably high even if k is small, we understand that 3-approximation of 2-changes is a great result. Thus meaning that the 5-approximation of 1-change algorithm is very strong considering the simplicity of the algorithm and it might prove usefull later on for the enhancement of other approximation results.

2.2.3 Maximum Leafy Forest

Following a small presentation of the second greedy algorithm the thought process and its results.

Problem Statement / Idea

The idea of this algorithm is finding leafy (many leaves) subtrees existing in a graph $G=(V,E)$ that combined make leafy forests as described by Lu and Ravi in [7]. The

last part is finding the maximum leafy forest. As a greedy algorithm depends on nodes of high degree. This algorithm wants to find the best iteration of a tree (maximal leaf tree) by joining these leafy subtrees into one and get the best approximation possible.

Preliminaries

Let G be a connected undirected graph. We use $V(G)$ to denote the set of nodes in G. We refer to an edge uw of G as the edge incident to u and w. For a subset of nodes S subset of V, let $\Gamma(S)$ denote the set of edges with exactly one endpoint in S. We sometimes overload notation and use $\Gamma(H)$ to denote $\Gamma(V(H))$ for a subgraph H of G. The degree of v in G is the number of edges of G incident to v. Let $V_i(G)$ denote the set of nodes that have degree i in G. Let $\bar{V}_i(G)$ denote the set of nodes that have degree at least i in G. Clearly $\bar{V}_0(G)=V(G)$. The leaves of G are the nodes in $V_1(G)$. A subtree of G is nonsingleton if it has more than one node.

Let T be a subtree of G.

We say T is leafy if $\bar{V}_3(T)$ is not empty, and every node in $V_2(T)$ is adjacent in T to exactly two nodes in $\bar{V}_3(T)$. A forest F of G is leafy if F is composed of disjoint leafy subtrees of G. We say F is maximally leafy if F is not a subgraph of any other leafy forest of G.

Algorithm

We give an approximation algorithm for Maximum Leaf Spanning Tree in this section. Given a graph G, our algorithm computes a spanning tree T for G by the following two steps. For obtaining a maximally leafy forest we have **Algorithm 3.**

1. Obtain a maximally leafy forest F for G.

2. Add edges to F to make it a spanning tree T for G.

Algorithm 3 *MaximallyLeafyForest*

Input : A graph $G=(V,E)$.

Output : F as a maximally leafy forest of G.

1: Let $F=\emptyset$.

2: **for** $\forall v \in G$ **do**

3: $S(v) \leftarrow v$

4: $d(v) \leftarrow 0$

5: **for** $\forall v \in G$ **do**

6: $S' \leftarrow \emptyset$

7: $d' \leftarrow 0$

8: **for** $\forall u$ *adjacent to* $v \in G$ **do**

9: **If** $u \notin S(v)$ and $S(u) \notin S'$ **then**

10: $d' \leftarrow d'+1$

11: Insert $S(u)$ into S'

12: **If** $d(v)+d' \geq 3$ **then**

13: **for** $\forall S(u) \in S'$ **do**

14: *Add edge* uv *to* F

15: *Union* $S(v)$ and $S(u)$

16: *Update* $d(u) \leftarrow d(u)+1$ and $d(v) \leftarrow d(v)+1$

Properties

Let F be a maximally leafy forest of G. Let $T_1 ... T_k$ be the disjoint leafy subtrees of F. One can verify that F has the following properties. We use the example in Figure (3) to illustrate each property. The dark lines in the figure are the edges in the maximally leafy forest F, which is composed of three leafy subtrees T_1, T_2, and T_3.

P1. Let w be a node in $\bar{V}_2(T_i)$. Then w cannot be adjacent in G to any node not in T_i. (Nodes v_1 and v_{27} are two examples of w e.g., suppose v_1 were adjacent to a node such as v_9, then F would not be maximal since the $e(v_1, v_9)$ could be added to F.)

P2. Let w be a node in T_i. Let w_1 and w_2 be two distinct nodes adjacent to w in G. If w_1 is not in F, then w_2 must be in T_i. (Nodes v_5 is an example of w If v_5 had two neighbors not in F, both these edges could be added to F contradicting its maximality.)

P3. Let w be a node not in F. If w is adjacent to two distinct nodes not in F then the degree of w in G is two. (Nodes v_9 and v_{15} are two examples of w Note that such nodes are not in F. If the degree of say v_5 were greater than

two, then v_5 and its three neighbors not in F could be added as an additional star in F contradicting its maximality again.)

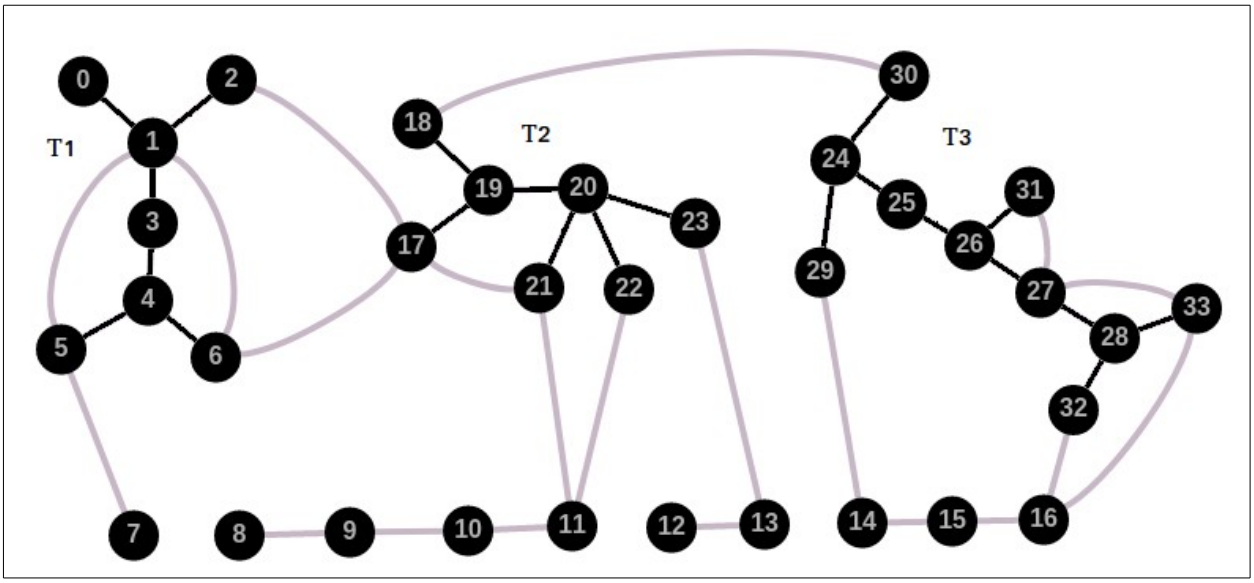


Figure (3): Properties of Maximally Leafy Forest

Results

Despite being greedy, the proof that by joining those leafy subtrees with their certain properties gives very good results as shown in Figure (4). The spanning tree T in dark lines is leafy, since every node in $V_2(T)$ is adjacent in T to exactly two nodes in $\bar{V}_3(T)$. Let the gray lines be the edges of G that are not in T. The maximum-leaf spanning tree of G is composed of all the edges incident to v_0 in G. Generalizing the example in a natural way yields examples in which the performance ratio of the algorithm asymptotically tends to 3. As suggested by the example shown in Figure (4), the approximation ratio of the algorithm might be improved by growing the maximally leafy forest by the descending order of the degree of nodes in G. We later use this suggestion in our algorithm for the MRCE problem with great results.

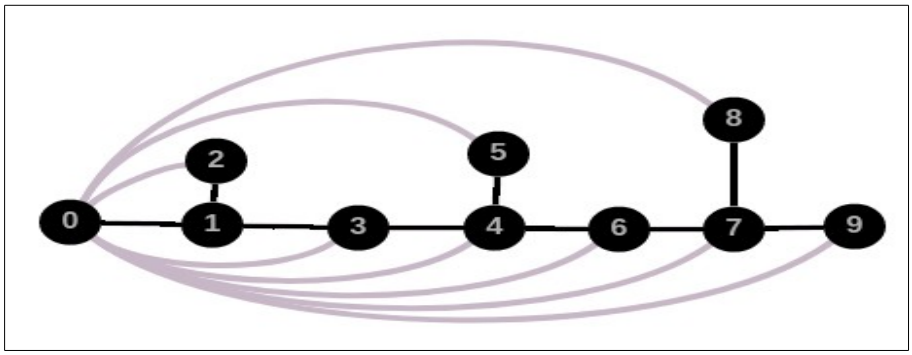


Figure (4): A Maximally Leafy Forest

2.3 Special Cases and Results

In this section we will explore some special cases of Connected Dominating Set and Maximum Leaf Spanning Tree.

2.3.1 Quota Steiner Tree

In this section we will shortly introduce the Quota Steiner Tree (QST) problem as a preliminary for the rest of the chapter, as it is used as a subroutine in most of the problems addressed later. Given a graph $G=(V,E)$, and a profit function p : $V \to \mathbb{Z} \cup \{0\}$ on the vertices , a cost function c : $E \to \mathbb{Z} \cup \{0\}$ on the edges and an integer q (quota).

The *Quota Steiner Tree* problem:

Find a subtree T that *minimizes* $\sum_{e \in E(T)} c(E)$, subject to $\sum_{v \in V(T)} p(v) \geq Q$.

Johnson et al. [8] studied the QST problem and showed that an α-approximation algorithm for the k-MST problem can be adapted to obtain an α-approximation algorithm for the Quota Steiner Tree problem. Using this result along with the 2-approximation for k-MST by Garg [9] results in a 2-approximation algorithm and we will consider it as a default algorithm denoted *QST*.

2.3.2 Budgeted and Partial Connected Dominating Sets

We now introduce two special cases of connected dominating set problem , their algorithms and results as presented by Khuller et al. in [10].

Budgeted Connected Dominating Set

In the budgeted connected dominating set problem (BCDS), given an undirected graph $G=(V,E)$, and an integer (budget) k, find a subset $S \subseteq V$ of at most k vertices such that the graph induced by S is connected, and the number of vertices dominated by S is maximized. **Algorithm 4** .

Partial Connected Dominating Set

In the partial connected dominating set problem (PCDS), given an undirected graph $G=(V,E)$ and an integer *quota* n', find a minimum size subset $S \subseteq V$ of vertices such the graph induced by S is connected, and S dominates at least n' vertices. **Algorithm 5** .

Algorithms

In broad lines, the algorithmic idea behind those two algorithms is to compute a greedy dominating set and its corresponding profit function and then obtain a connected subgraph via an approximation algorithm for the Quota Steiner Tree problem with different elements as an input. We formally present the algorithms of the previous problems and the algorithm for finding a greedy dominating set *(GDS)* **Algorithm 4**.

Algorithm 4 *Greedy Dominating Set*

Input: Graph $G=(V,E)$.

Output: Dominating Set D and profit function $p : V \to \mathbb{Z} \cup \{0\}$.

1: $D \leftarrow \emptyset$

2: $U \leftarrow V$

3: **for all** $v \in V$ **do**

4: $p(v) \leftarrow 0$

5: **end for**

6: **while** $U = \emptyset$ **do**

7: $v \leftarrow argmax_{v \in V \setminus D} |N_U(v)|$ → $N_U(v)$ *is the set of neighbors of* v , *including itself, in the set* U

8: $C_v \leftarrow N_U(v)$

9: $p(v) \leftarrow |C_v|$

10: $U \leftarrow U \setminus N_U(v)$

11: $D \leftarrow D \cup \{v\}$

Algorithm 5 *Greedy Profit Labeling Algorithm for PCDS*

Input: Graph $G=(V,E)$ and $n' \in \mathbb{Z} \cup \{0\}$.

Output: Tree T with at least n' Coverage.

1: Compute the greedy dominating set D and the corresponding profit function $p: V \to \mathbb{N}$ using the **Algorithm 4**

2: Use the 2-approximation for QST problem [9] on the instance (G,p) to obtain a tree T with profit at least n'

Algorithm 6 *Greedy Profit Labeling Algorithm for BCDS*

Input*:* Graph $G=(V,E)$ and $k \in \mathbb{N}$.

Output*:* Tree $\widetilde{T}$ with cost at most k.

1: Compute the greedy dominating set D and the corresponding profit function $p: V \to \mathbb{N}$ using the **Algorithm 4**

2: $Opt \leftarrow$ number of vertices dominated by an optimal solution ➔ Guess using binary search between k and n

3: Use the 2-approximation for QST problem [9] to obtain a tree T with profit at least $(1-\frac{1}{e})Opt$

4: Use the **dynamic program** to find $\widetilde{T}$,the best subtree of T having at most k vertices

Results and discussion

As we can see these algorithms give for the Partial Connected Dominating Set problem a $4\ln\Delta+2+o(1)$ - approximation and for Budgeted Connected Dominating Set a $\frac{1}{13}(1-\frac{1}{e})$ - approximation. We can see that those problems are modified to be reduced in a Quota Steiner Tree problem and be solved that way ,which as we will discuss later is a very well know approach in those special cases of connected dominating sets. This reduction into QST will always give good approximation ratios but our question is how much of an approximation trade off is there due to the transposition of those problems. The question we ask ourselves is, if we can find improved greedy or local algorithms for those problems with better approximation.

2.3.3 K-Minumum Spanning Tree and Prize Collecting Steiner Tree

We now introduce two special cases problems of minimum spanning trees, their connection and results. We will also present the outline of k-MST algorithm. Both of this algorithms will be addressed for their rooted versions.

Prize Collecting Steiner Tree (Goemans-Williamson Minimization Problem)

Given a graph $G=(V,E)$, a non-negative edge cost $c(e)$ for each edge $e \in E$, a non-negative vertex prize $p(v)$ for each vertex $v \in V$, and a specified root vertex $v_0 \in V$. We will now consider the optimization problem based on this scenario called the *Goemans-Williamson Minimization problem :*

Find a subtree $T'=(V',E')$ of G that minimizes the cost of the edges in the tree plus the prizes of the vertices not in the tree, i.e., that minimizes :

$$GW(T')=\sum_{e\in E'} c(e)+\sum_{v\notin V'} p(v)$$

In [11], Goemans and Williamson present an $O(n^2\log n)$ time primal-dual approximation algorithm for the rooted version of GW-Minimization that is guaranteed to be within a factor of $2-\frac{1}{(n-1)}$ of optimal, where $n=|V|$.

K-Minumum Spanning Tree

Given an undirected graph $G=(V,E)$ with nonnegative edge costs and an integer k, the k-MST problem is that of finding a tree of minimum cost that spans k vertices of G. We refer to a tree that spans k vertices as a k-tree. Note that we may assume that the edge costs satisfy the triangle inequality without loss of generality. The k-MST problem was shown to be NP-hard by Ravi et al. in [12]. The rooted version of the k-MST problem requires inclusion of a specific root node in the k-tree. As observed in [13], solving the rooted and unrooted versions are essentially equivalent. As we discussed previously Garg gave a 2-approximation algorithm for the (rooted) k-MST problem based on the Goemans-Williamson Prize-Collecting Steiner Tree approximation algorithm. Johnson et al. showed in [8] that any polynomial time α-approximation algorithm for (rooted) k-MST, which applies GW, yields a polynomial time α-approximation algorithm for (rooted) QST. Derived from their work we know that there is a 2-approximation algorithm for Rooted Quota Steiner Tree (RQST) and we will name it *2-RQST* . We present the outline of an algorithm to the rooted version of the problem for simplicity.

Algorithm

We used the definitions of the *GW-minimization problem* and *Prize Collecting Steiner Tree* (PCTS) as preliminaries because they reduced the k-MST problem to the PCST and used the GW-minimization algorithm to solve it. We now present the outline **Algorithm 7** for the k-MST problem whilst further definitions our found in [11].

Algorithm 7 *Goemans-Williamson minimization for the k-MST rooted problem*

Input: Graph $G=(V,E)$ with edge cost $c_e \geq 0$, a root r, the cost L of an optimal k-tree containing r , and a fraction a.

Output: Tree F' containing r and at least ak nodes.

Discussion

As we have previously seen, many problems are being transposed to the *Quota Steiner Tree* problem and its different cases. K-MST used a format of a *Prize Collecting Steiner Tree* and *GW-minimization* algorithm that solves a special problem for this format. For the same reason Sigalas et al. have used the *GW-minimization* algorithm since they have reduced the problem of MRCE to a QST problem. As we have seen already many problems have been reduced to QST problems due to the lack of good quality approximation local and greedy algorithms derived from the MLST and CDS problems.

2.4 Related Combinatorial Optimization Problems

In the last section of this chapter we will formally present the connection between CDS and MLST and a hard problem named Max Weight Budgeted Connected Set Cover (MWBCSC) that uses the knowledge of specialized algorithms used on both of those problems special cases. We will finally discuss the approach of combining algorithms for enhanced optimization which is used regurarly in this field of research.

2.4.1 Connected Dominating Set and Maximum Leaf Spanning Tree

In this section we will formally introduce the important connection between those problems and its proof. We will also re-address to the CDS problem to explain the significance of the Connected Domination Number as described in [14].

Connected Domination Number

The *connected domination number* of a graph G is the minimum cardinality of CDS, denoted by $\gamma_c(G)$. A CDS that has size equal to the domination number is called a *minumum* CDS. The *connected domination number* is a classical subject studied in graph theory for many years. Some interesting results are obtained in those ealier efforts. The following is one example.

Let $\lambda(G)$ denote the *max leaf number* of a graph G , which is the maximum number of leaves in a spanning tree T_s of G.

We present next theorem :

For any graph G *of order* n , $\gamma_c(G)=n-\lambda(G)$.

We present the **proof** of the previous theorem :

It is easy to see that for any tree T_s , $\gamma_c(T_s)=|V(T_s)|-\lambda(T_s)$. Moreover, a CDS for a spanning tree T_s of G is also a CDS for G. Therefore, $\gamma_c(G)\leq n-\lambda(G)$. Now, consider a minimum CDS D of G. Let H be a spanning tree of $G[D]$ where $G[D]$ is the subgraph of G induced by D. Connect H to every vertex in $V-D$ to obtan a spanning tree T_s of G. Then, every vertex in $V-D$ is a leaf of T_s . Conversly, every leaf of T_s is in $V-D$. Otherwise, if T_s has a leaf x not in $T-D$, then $D-\{x\}$ would be a CDS for T_s and hence a CDS for G , contradicting the minimality of D.

2.4.2 Max Weight Budgeted Connected Set Cover

Let us discuss now the approximation algorithm for the maximum weight budgeted connected set cover (MWBCSC) problem as presented in [15]. Given an element set X , a collection of sets $\boldsymbol{S}\subseteq 2^X$, a weight function w on X , a cost function c on $\boldsymbol{S}$, a connected graph $G_{\boldsymbol{S}}$ (called communication graph) on vertex set $\boldsymbol{S}$,and a budget L , the MWBCSC problem is to select a subcollection $\boldsymbol{S}'\subseteq S$ such that the cost $c(\boldsymbol{S}')=\sum_{S\in\boldsymbol{S}'} c(\boldsymbol{S})\leq L$, the subgraph of $G_{\boldsymbol{S}}$ induced by $\boldsymbol{S}'$ is connected, and the total weight of elements covered by $\boldsymbol{S}'$ (that is $\sum_{x\in S_{S\in\boldsymbol{S}'}} w(x)$) is maximized. We present a polynomial time algorithm that its performance ratio is $O((\delta+1)\log n)$, where δ is the maximum degree of the communication graph $G_{\boldsymbol{S}}$ and n is the number of sets. This ratio can be improved to $O(\log n)$ if every set has a cost at most half of the budget $L/2$. We will provide the set of algorithms for the solution of this problem and later we will discuss which was their thought process on approacing this problem. We omit defining how does profit and cost functions work due to their relation with another mathematical subject called *submodularity*.

Algorithm

Here we formally describe the **Algorithms 8,9,10** used to solve this problem.

Algorithm 8 *Algorithm Defining Profit*

Input: An instance (G,c,L).

Output: A (complete) set cover A and a profit function π on $\boldsymbol{S}$.

1: $A \leftarrow \emptyset, F \leftarrow X$

2: **for** $S \in \boldsymbol{S}$ **do**

3: $\pi(S) \leftarrow 0$

4: **while** $F \neq \emptyset$ **do**

5: *Select* $S \in \boldsymbol{S} \setminus A$ that *maximizes* $\Delta_S w(A)/c(S)$

6: $\pi(S) \leftarrow \Delta_S w(A)$ (marginal profit of S over A)

7: $A \leftarrow A \cup \{S\}, F \leftarrow F \setminus F(S)$

Algorithm 9 *Selection of* B

Input: The collection of sets A, cost function $c: A \rightarrow R$, profit function $\pi: A \rightarrow \mathbb{R}$ and budget L.

Output: A subcollection of sets $B \subseteq A$.

1: $R \leftarrow 0, B \leftarrow \emptyset, i \leftarrow 1$

2: **while** $R + c_i \leq L$ and $A \neq \emptyset$ **do**

3: **if** $S_i \cap F(OPT) \neq \emptyset$ **then**

4: $B \leftarrow B \cup \{S_i\}$, $R \leftarrow R + c_i$

5: $i \leftarrow i+1, A \leftarrow A \setminus \{S_i\}$ (removes set S_i from A, cause we added it into B)

8: Let $S_{max} = arg\,max\{\pi(S): S \in \boldsymbol{S}\}$

9: **if** $\pi(B) < \pi(S_{max})$ **then**

10: $B \leftarrow S_{max}$

Algorithm 10 *Main Algorithm For MWBCSC*

Input: An instance (X, S, w, c, G_S, L).

Output: A collection of sets $\boldsymbol{S}' \subseteq S$ with cost at most L such that $G_S[\boldsymbol{S}']$ is connected.

1: Using **Algorithm 8** to assign profit π on the sets of $\boldsymbol{S}$.

2: Apply an α-approximation algorithm on instance (G_S, c, π, L) to obtain a budgeted node weighted Steiner tree T. Output $\boldsymbol{S}' = V(T)$.

2.4.3 Thoughts on previous algorithms

In computational complexity theory, a reduction is an algorithm for transforming one problem into another problem. A sufficiently effiecient reduction from one problem to another may be used to show that the second problem is at least as difficult as the first.

Intuitively, problem A is **reducible** to problem B if an algorithm for solving problem B efficiently (if it existed) could be used as a subroutine to solve problem A efficiently. When this is true, solving A cannot be harder than sovling B. “Harder” means having a higher estimate of the required computational resources. We can also use this property if problem A does not have efficient approximation algorithms for its solution.

Since we introduced the connection between Connected Dominating Set and Maximum Leaf Spanning Tree we understand that the algorithms built to solve those problems are entirely similar or have some steps that may be the same. As we have noticed CDS can be solved with a local approach 1-hop and MLST with 1-LOT. These approaches are so closely related to the local optimality that make us think that those problems could be solved the same way.

As we said in previous discussions in this chapter, we can **transpose** a part of the problem A to another B and use the respective algorithm of B as a subroutine to A . Transposing means that we change the structure of a problem A into another B without changing neither the input nor the output. A great example is the Budgeted and Partial Connected Domiting Sets that use the Quota Steiner Tree algorithm as the second subroutine and applying it in a greedy dominating set obtained by the first subroutine. This partial transposition gives more efficient approximation ratio than trying to re-structure the initial algorithms used for CDS problems i.e 1-hop into solving BCDS. We further explain that in general when we solve a problem A with its respective algorithm X there is a propabiliy of improving or restructuring X into giving better results. However what we presented in this chapter shows that when we divide problem A into B and C which are solved by the combination of Y and Z respectively then we usually get better results than X. There is also the inability to solve efficiently special case problems with the main general algorithms. Suppose we have a newly defined problem D is a special case of A i.e BCDS of CDS. We can see that restructuring X solving A cannot effieciently solve BCDS. Since we have this issue we suggest that transposing problems is a great way of bypassing the general algorithms of all kinds (local, greedy) into new combinatorial algorithms. Our goal and research is to create a specified approach into using this tool of transpozing problems and combining different algorithms so as to optimize and solve old and newly defined problems.

Considering the MLST we can see that there is a silver lining between greediness and local optimality as we have presented previously in this chapter. The results are very good and unfortunately are not yet being used as subroutine in transposed problems,

but they are used as heuristics on top other algorithms results. Nontheless we later in chapter 3 are using the idea of greedy leafy forest for our algorithm and we suggest of using the 1-LOT, as a heuristic. The idea of heuristics is thouroughly analyzed in [16].

Now as the combination or selection of algorithms that solve other cases occurs as an approach in solving problems with “similar” difficulty , we can see that the k-Minimum Spanning Tree is broken into parts and solved by other problems : GW-minimization on a Prize Collecting Steiner Tree. This means that we can bypass the whole internal approach of solving a problem if we can transpose every part of it into other problems more effieciently solved. The point is we have to research and have knowledge of the little intricacies so as to be able to do that without loss of quality in our results due to this transpositison. Finally we have addressed the Max Weight Budget Connected Set Cover problem which is the finest approach to solving a problem by considering every part of it as a different problem. The peek of their research is that in their attempt to solve it this way they were forced to remodel some of the algorithms and problems. The explanation of their thought process in their paper is of great significance in studying and trying to contribute into the domain of graph theory and its problems.

3. MAXIMUM ROOTED CONNECTED EXPANSION

We now define the Maximum Rooted Connected Expansion problem its related work, algorithm, ideas and our contribution.

3.1 Maximum Rooted Connected Expansion

We now present the thought process , algorithms and results of Lamprou et al. for the MRCE problem.

3.1.1 Problem Statement

Given a graph G and a root node u_0 , we wish to find a subset of vertices S such that S is connected, S contains u_0 and the ratio $N[S]/|S|$ is maximized, where $N[S]$ denotes the closed neighbourhood of S , that is $N[S]$ contains all nodes in S and all nodes with at least one neighbour in S.

Problem Statement 3 (Maximum Rooted Conncected Expansion, MRCE)

Input : A graph $G=(V,E)$ and a root v_0.

Output : The MRCE number : $MRCE(G,v_0)=\max_{S\in con(G,v_0)} \frac{|N[S]|}{|S|}$,

where $con(G,v_0)\leftarrow\{S\subseteq V(G)\}$ where $v_0\in S$ and S is connected.

3.1.2 Preliminaries

A graph G is denoted as a pair $(V(G),E(G))$ of the nodes and edges of G. The graphs considered are simple (neither loops nor multi-edges are allowed), connected and undirected. Two nodes connected by an edge are called adjacent or neighboring. The open neighborhood of a node $v\in V(G)$ is defined as $N(v)=\{u\in V(G):\{v,u\}\in E(G)\}$, while the closed neighborhood is defined as $N[v]=\{v\}\cup N(v)$. For a subset of nodes $S\subseteq V(G)$, we expand the definitions of open and closed neighborhood as $N(S)=N(v)_{v\in S}\setminus S$ and $N[S]=N(S)\cup S$. The degree of a node $v\in V(G)$ is defined as $d(v)=|N(v)|$. The minimum (resp. maximum) degree of G is denoted by $\delta(G)=min_{v\in V(G)}d(v)$ (resp. $\Delta(G)=max_{v\in V(G)}d(v)$.

3.1.3 Algorithm

Their algorithm closely follow the work in [10] for the BCDS problem. In broad lines, their algorithmic idea is to compute a greedy dominating set and its corresponding profit function and then obtain a connected subgraph via an approximation algorithm for the

QST problem. Evidently, both MRCE and BCDS require finding a connected subset $S \subseteq V(G)$ with many neighbors. Nonetheless, while in BCDS we only care about maximizing $|N[S]|$, in MRCE we care about maximizing $\frac{|N[S]|}{|S|}$ with the additional demand that $v_0 \in S$. In order to deal with this extra requirement, they also addressed the rooted version of QST, namely the Rooted Quota Steiner Tree (shortly RQST) problem. **Algorithm 4**, namely the Greedy Dominating Set (shortly GDS) algorithm, describes a greedy procedure to obtain a dominating set and a corresponding profit function for the input graph G. At each step, a node dominating the maximum number of the currently undominated vertices is chosen for addition into the dominating set. **Algorithm 11**, namely the Greedy MRCE algorithm, makes use of GDS to obtain a dominating set for a slightly modified version of G, namely a graph G , which is the same as G with the addition of n^2 leaves to node v_0. Then, the algorithm outputs a connected subset T_i (containing v_0) for any possible size i. Finally, the subset yielding the best MRCE ratio is chosen as the approximate solution. In terms of notation, we refer to the approximation **Algorithm 7** as the *2-RQST* (G, v_0, p, q) algorithm with a graph G, a root node $v_0 \in V(G)$, a profit function $p: V(G) \to N \cup \{0\}$ and a quota q as input. We omit including an edge cost function, since in our case all edges have the same cost, that is, cost 1. Furthermore, let $[n] \leftarrow \{1,2,3,\ldots,n\}$.

Now, consider a connected set S_i of size i (which contains v_0) yielding the maximum number of dominated vertices, i.e. $S_i \in argmax_{S: S \in con(G, v_0), |S|=i} |N[S]|$. We then denote $OPT_i \leftarrow |N[S_i]|$ and use it in the quota parameter of 2-RQST at line 4 of Greedy MRCE. Yet, in the general case, we do not know OPT_i and also such a quantity may be hard to compute. To overcome this obstacle, notice that $OPT_i \in [i, n]$ and therefore we could guess OPT_i, e.g., by running a sequential or binary search within the loop of Greedy MRCE and then keeping the best tree returned by *2-RQST*. Notice that such an extra step requires at most a linear time overhead. Therefore, the running time of Greedy MRCE remains polynomial and is dominated by the running time of *2-RQST*. For presentation purposes, we omit this extra step and assume OPT_i is known for each $i \in [n]$.

Algorithm 11 *Greedy MRCE*

Inputx: A graph plus node pair (G, v_0).

Output: An MRCE solution S and its corresponding ratio R ,

where (MRCE solution = set S , ratio = number R).

1: *Construct* G' : same as G with extra n^2 leaves attached to v_0

2: $(D, p) \leftarrow GDS(G')$ {**Algorithm 4** Greedy Dominating Set}

3: **for** $\forall i \in [n]$ **do**

4: $T_i \leftarrow 2-RQST(G, v_0, p, (1-\frac{1}{e})OPT_i)$ {**Algorithm 7** GW}

5: *Let* $i_s = argmax_{i \in [n]} \frac{|N[T_i]|}{|T_i|}$

6: $S \leftarrow T_{i_s}$, $R \leftarrow \frac{|N[T_{i_s}]|}{|T_{i_s}|}$

7: **return** S , R

3.1.4 Results

Lamprou et al. have studied MRCE for several graph cases , interval , split and general. In this section we only analyzed their algorithm for general graphs. They gave a constant-factor approximation algorithm by exploring the relation of MRCE with BCDS [15] which gives a $\frac{1}{6}(1-(\frac{1}{e}))$ -approximation. The major open question is to improve the approximability of the problem on general graphs without applying BCDS techniques, but using rather MRCE properties.

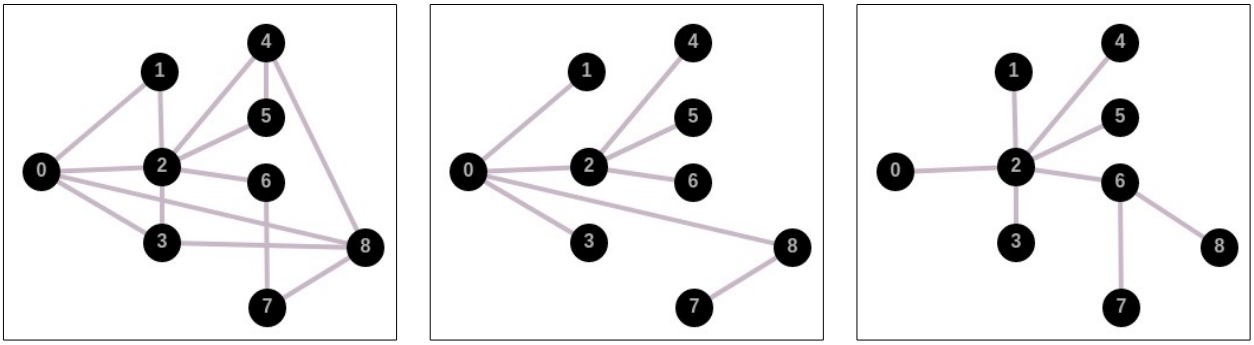
3.2 Our Contribution

In this section we explain our thought process on how we constructed the algorithm, our experimental results and our intuition into providing even better approximation.

3.2.1 Approaching the problem

Since our study on local search approximation algorithms for CDS and MLST, *1-hop* and *1-LOT* respectively, we considered applying them to our problem due to their good approximation ratios. Furthermore, due to the fact that MRCE requires a root node v_0 we intuitevely thought that local search will give even better approximation ratio. Our belief was that since we already have the root node of our set S ,the local search to

find the best S would be easier by just expanding it step by step into more distant neighborhoods if they would improve the MRCE number (ratio). Since we have seen transposing problems into tree problems i.e QST is easily modeled and providing good results, we decided to apply *1-LOT*. However in order to apply *1-LOT* we were forced to find a Maximum Leaf Spanning Tree T_{OPT}, without disconnecting nodes from v_0. Then we should try and find a subset S of this T_{opt} where $v_0 \in S$. Unfortunatelly we have observed that it is not necessary to find a T_{opt} in order to find a subset S with better ratio and we provide a proof of non-existance through a contradiction example shown in Figure (5) below. Our assumption is that the $MRCE_{OPT}$ is not necessarily found in $MLST_{OPT}$.



(G) **(T1)** **(Topt)**

Figure (5): Proof of non-existance

Proof of non-existance

Considering T_1 an arbitrary spanning tree of G and T_{OPT} a maximum leaf spanning tree of G. We can easily see that if v_0 is our root node and the best set $S=\{v_0, v_2\}$ in T_1 we get $R_{T_1}=8/2$, where $R_{T_1}=\frac{|N[S]|}{|S|}$ and with the same S in T_{OPT} we get $R_{T_{OPT}}=7/2$, thus $R_{T_1} > R_{T_{OPT}}$ proving our assumption.

We then considered the greedy approach on trees utilizing the *Max Leafy Forest* properties and algorithm. Our thought was to find leafy subtrees and connect them with the subtree of v_0, T_{v_0}. Although the properties of leafy subtrees were very important we could prove a good approximation ratio only for a maximum leaf spanning tree, which for MRCE did not provide an approximation to the $MRCE_{OPT}$ solution as we proved previously. However we kept the idea of leafy subtrees in our algorithm as we are iteratevily searching for high degree nodes that we handle them as stars.

3.2.2 Algorithm

Since our algorithm had to be tweaked to handle high degree nodes and not leafy subtrees, we considered *labeling* as it was modeled in Budgeted problems by a profit function p and applied on all nodes in G in order to connect highest degree nodes with v_0 resulting in a good ratio R. However *labeling* is pretty costly in approximation and we managed to bypass it by the iterative property of the algorithm. We will now present some preliminaries and then formally present our **Algorithm 12**. We will finally discuss each step and its respective properties that cumulative improve our approximate solution.

Preliminaries

Early in our algorithm and in every iteration we create a set which we denote a set of high degree nodes $H \subseteq G(V)$.

We also want to check the MRCE number otherwise ratio denoted by $R = \frac{|R[S]|}{|S|}$.

Set H is handled as a list. Furthermore , when we denote H *not traversed*, we mean that the list iteration of set H has not ended. Also, when we add a node v_i back in H it goes on the end of the list of H.

Finally *shortest-path* function with (v_i, S) as input, finds the $path$ between v_i and a node $u_i \in S$ with minimum distance.

Algorithm

We will now describe the final version of our algorithm.

Algorithm 12 *Greedy Iteration MRCE*

Input:A graph and a node pair (G, v_0).

Output: An MRCE solution S and its corresponding ratio R.

1. $S \leftarrow \emptyset$, $H \leftarrow \emptyset$
2. $R_{max} \leftarrow R_S$, $S \leftarrow \{v_0\}$
3. *Find* $H = \{v_i, \ldots, v_k\}$, where $d(v_i) \geq R_{max}$
4. **while** H *not traversed* **do**
5. *shortest-path* (v_i, S)
6. $H \leftarrow H - \{v_i\}$ (*remove* v_i from H)
7. **if** $R_{S \cup v_i} > R_{max}$ **then**

8. $R_{max} \leftarrow R_{S \cup v_i}$, $S \leftarrow S \cup \{v_i\}$
9. $H \leftarrow H - \{x\}$ (*remove* x from H) , where x node with $d(x) < R_{max}$
10. **else**
11. *add* v_i in the end of the list H
12. **end while**
13. **if** $R_s < R_{max}$ **then**
    $R_S \leftarrow R_{max}$, **repeat** from **line 2**
14. **else**
15. **return** S , R_{max}

We will shortly analyze our algorithm through previous ideas. We kept the leafy subtree idea by finding in every iteration the high degree nodes. Those high degree nodes are conceived as leafy singleton subtrees. We connect all those high degree nodes with our greedy connection function shortest-path. Despite shortest path being greedy and should generally give us overall worst approximation, it has a great quality. Shortest-path finds a minimum distance connected dominated set S. Minimizing set S means that our denominator is as small as possible thus improving R. Our nested iteration handles H and tries to connect high degrees in H with v_0. This alone is being greedy as if we were only labeling our nodes thus not necessarily giving better approximation than just labeling. However in combination with the outer iteration we get a new H that omits reduntant, lower degree nodes, giving better and better approximation in every iteration. Basically our concept is to continuously increase the lower bound of what we call high degree nodes until we cannot profitably connect them to v_0 in comparison with its previous iteration. Due to shortest-path function we can see that set S includes previous iterations high degree nodes, despite of not being neither the source S_{v_0} nor goal $v_i \in H$ variables of the function. Our experimental analysis gave us great examples of what we are currently presenting.

3.2.3 Results and Future Work

We have implemented a program to test our algorithm and make our expiremental analysis. Our expiremental analysis suggests a 3-approximation algorithm solving the MRCE problem. We have tested our algorithm in various cases of general graphs considering high, medium, low density graphs. We then experimented with different size of datasets where number of nodes in G is n, considering bulk $n \geq 1000$, medium

$150<n\leq 400$, and small $10<n<50$. Considering future work we are currently working on the theoretical analysis of our algorithm which suggests that our algorithm gives at least a 3-approximation ratio. The outline of the analysis is that if we could connect $S=\{v_0\}$ denoted S_{v_0} with other neighboring nodes of high degree that do not have common neighbors with eachother then the final set would be optimal thus giving us optimal ratio R_{OPT}. In reality we have to connect this set with distant high degree nodes with an appropriate path and then we would gain the optimal ratio R_{OPT}. However we connect S_{v_0} with $v_i \in H$ with *shortest-path* which is not the appropriate path. We denote the appropriate path *bulkiest-path* defined by its set S_B. We assume that the length of *bulkiest-path* is equal to the length of *shortest-path* +1, but the nodes included in S_B are of the highest possible degree, meaning they have greater degree than those in *shortest-path* set S_S resulting in a greater ratio R_{OPT} despite $|S_B|=|S_S|+1$. Our algorithm gives $R_S=\frac{|N[S_S]|}{|S_S|}$ and we assume that the optimal solution is $R_B=\frac{|N[S_B]|}{|S_B|}=R_{OPT}$. However we are very close to prove that $\frac{R_S}{R_B}\geq ½$. That means we hope we can prove a 2-approximation algorithm.

We also have an idea of implementing 1-LOT as a heuristic on top of our results of Greedy Iteration MRCE which as we have tested experimentally might give a small improvement in this 2-approximation expected algorithm. Generally as we have studied [16], by adding heuristics in those kind of problems results in small improvements in approximation.

4. CONCLUSIONS

The main purpose of this thesis was to conduct a survey on some Graph Theory problems and analyze them. One of these problems was the newly defined Maximum Rooted Connected Expansion that is of great practical importance in data prefetching. Since the other problems have been extensively analyzed by the research community and have greatly refined algorithms for their approximate solution as we presented here, we were able to contribute to the MRCE with new ideas and an algorithm.

In the first part of the thesis we presented many important Graph Theory problems by explaining their idea behind their solution, their algorithms and results. We did that as a roadmap for the readers so as to understand the basics of the problems and their connection between them. We formally presented their algorithms in order to explain how their techiniques can be used into eachother. Despite being presented as a study of CDS and MLST problems and their special cases, the presentation is unfolded in a very specific way so that the reader would follow our thought process into creating a new algorithm for a different problem. Most of the ideas behind their connection are mentioned at the end of each respective section and others possibly occur while studying this thesis thoroughly. We also presented the MWCSC problem as an example of a study that utilizes the structures and algorithms of the aformentioned problems to solve this.

The second part was a formal presentation of the main subject of the thesis the MRCE problem. We thoroughly described how the problem was defined and its practical application. We then describe our algorithm and how it follows a completely different path than the one Lamprou et al. followed. We lightly compared those two algorithms suggesting our algorithm gives better approximation, but our main purpose was to introduce new ways of approaching the problem.

Finally, apart from the theoretical standpoint, which we considered, we must not surpass the effect of the algorithms on applications. For that reason although we did extensive theoritical analysis in this thesis, we also implemented the algorithm so as to give some experimental results. Our purpose was to suggest that we need to formally implement as many of those problems as possible so as to compare them and their results in a more practical way. The results could lead to useful conclusions and be of use to the respective research communities.

REFERENCES

[1] F.V.Fomin, F.Giroire, A. Jean-Marie, D.Mazauric, N.Nisse, To Satisfy Impatient Web surfers is Hard, 2012.

[2] I.Lamprou, R.Martin, S.Schewe, I.Sigalas, V.Zissimopoulos, Maximum Rooted Connected Expansion, 2018.

[3] S.Guha and S.Khuller, Approximation algorithms for connected dominating sets. Algorithmica, 374–387, 1998.

[4] S.Khuller and Yong, Revisiting Connected Dominating Sets:An Optimal Local Algorithm, 2001.

[5] M.R.Garey, D.S.Johnson, Computers and Intractability: A guide to the theory of NP-completeness, 1979.

[6] R.Ravi and H.-I.Lu, The Power of Local optimization, 1996.

[7] R.Ravi and H.-I.Lu, Approximating Maximum Leaf Spanning Trees in almost linear time, 1998.

[8] D.S.Johnson, M.Minkoff and S.Phillips, The prize collecting Steiner tree problem: theory and practice, SODA, 760–769, 2000.

[9] N.Garg, Saving an epsilon: a 2-approximation for the k-MST problem in graphs, STOC, 396–402, 2005.

[10] S.Khuller, M.Purohit and K.K.Sarpatwar, Analyzing the Optimal Neighborhood: Algorithms for Budgeted and Partial Connected Dominating Set Problems, 2013.

[11] M.X.Goemans and D.P.Williamson. The primal-dual method for approximation algorithms and its application to network design problems, 1997.

[12] R.Ravi, R.Sundaram, M.V.Marathe, D.J.Rosenkrantz, and S.S.Ravi, Spanning trees short and small, 1994.

[13] B.Awerbuch, Y.Azar, A.Blum, and S.Vempala, Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen, 1995.

[14] D.-Z.Du and P.-J. Wan, Connected Dominating Set, Theory and Applications, Springer.

[15] Y.Ran, Z.Zhang, K-I. Ko and J.Liang, An approximation algorithm for maximum weight budgeted connected set cover, 2015.

[16] Y.Wang, S.Cai, and M.Yin, Two Efficient Local Search Algorithms for Maximum Weight Clique Problem , 2016.