MINIMUM HUB COVER PROBLEM: SOLUTION METHODS & APPLICATIONS

by

BELMA YELBAY

Submitted to the Graduate School of Engineering
and Natural Sciences in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Sabancı University

Summer 2014

MINIMUM HUB COVER PROBLEM:
SOLUTION METHODS & APPLICATIONS

by Belma Yelbay

APPROVED BY:

Prof. Ş. İlker Birbil ..... (Thesis Advisor)

Assoc. Prof. Kerem Bülbül ..... (Thesis Advisor)

Assist. Prof. Tınaz Ekim Aşıcı .....

Assoc. Prof. Özgür Gürbüz .....

Assoc. Prof. Tonguç Ünlüyurt .....

DATE OF APPROVAL: August 6, 2014

To my lovely son and husband...

MINIMUM HUB COVER PROBLEM:
SOLUTION METHODS & APPLICATIONS

Belma Yelbay

PhD Thesis, 2014

Thesis Advisors:
Prof. Ş. İlker Birbil
Assoc. Prof. Kerem Bülbül

Keywords: minimum hub cover problem, graph query processing, graph covering problem

The minimum hub cover is a new $\mathcal{NP}$-hard optimization problem that has been recently introduced to the literature in the context of graph query processing on graph databases. The problem has been introduced as a new graph representation model to expedite graph queries. With this representation, a graph database can be represented by a small subset of graph vertices. Searching over only that subset of vertices decreases the response time of a query and increases the efficiency of graph query processing. We introduce the problem of finding a subgraph including the minimum number of vertices as an optimization problem referred to as the minimum hub cover problem. We demonstrate that searching a query over the vertices in minimum hub cover increases the efficiency of query processing and surpasses the existing search methods.

We also introduce several mathematical programming models. In particular, we

give two binary integer programming formulations as well as a novel quadratic integer programming formulation. We use the linear programming relaxations of the binary integer programming models. Our relaxation for the quadratic integer programming model leads to a semidefinite programming formulation. We also present several rounding heuristics to obtain integral solutions after solving the proposed relaxations.

We also focus on planar graphs which have many applications in planar graph query processing and devise fast heuristics with good solution quality for minimum hub cover. We also study an approximation algorithm with a performance guarantee to solve the minimum hub cover problem on planar graphs. We conduct several numerical studies to analyze the empirical performances of solution methods proposed in this thesis.

EN KÜÇÜK GÖBEK KAPSAMA PROBLEMİ: ÇÖZÜM METODLARI VE UYGULAMALAR

Belma Yelbay

Doktora Tezi, 2014

Tez Danışmanları:
Prof. Dr. Ş. İlker Birbil
Doç.Dr. Kerem Bülbül

Anahtar Kelimeler: en küçük göbek kapsama problemi, çizge sorgu işleme, çizge kapsama problemi

En küçük göbek kapsama problemi, çizge veri tabanları üzerinde sorgulama alanında literatüre kazandırılmış yeni bir eniyileme problemidir. Problem çizge sorgularının hızlandırılması için yeni bir çizge gösterim şekli olarak tanıtılmıştır. Bu gösterim şekli ile bir çizge veritabanı o çizginin düğümlerinin bir alt kümesi cinsinden ifade edilmektedir. Bu alt küme üzerinden sorgu işlemek, sorgu süresinin azalmasını ve sorgu sürecinin verimliliğinin artmasını sağlamaktadır. Bu çalışmada en az sayıda düğümle bir çizgeyi ifade etme problemini en küçük göbek kapsama problemi olarak tanımlıyoruz. Bu alt küme üzerinden sorgulama işlemi yapılmasının sorgu verimliliğini artırdığını ve mevcut yöntemlere göre avantaj sağladığını gösteriyoruz.

Bu çalışmada en küçük göbek kapsama problemi içlerinde 0-1 tamsayılı programlama formülasyonu ile ikinci dereceden tamsayılı programla modelinin de olduğu farklı matematiksel programlama modelleri tanıtıyoruz. İkinci dereceden tamsayılı program-

lama modelinin gevşetilmesini kullanarak yarıbelgili programlama formülasyonu elde ediyoruz. Doğrusal ve yarıbelgili gevşetme modellerini kullanan yuvarlama yöntemleri önererek bu yöntemlerin deneysel performanslarını karşılaştırıyoruz.

Ayrıca düzeysel çizge veritabanlarında obje tanımlama, biyometrik kimlik belirleme gibi sorgu işleme uygulamaları olan düzeysel çizgelere odaklanıp iyi çözümler üreten hızlı sezgisel geliştiriyoruz. En küçük göbek kapsama problemini düzeysel çizgeler üzerinde performans garantisi veren bir yaklaşıklama algoritması tanıtıyoruz. Ayrıca yaklaşıklama algoritmasının teorik performansını ispatlayarak, kapsamlı deneysel bir çalışma ile algoritmanın deneysel performansını ölçüyoruz. Ayrıca bir çok deneysel çalışma ile çözüm yöntemlerimizin deneysel performanslarını ölçüyoruz.

Acknowledgments

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

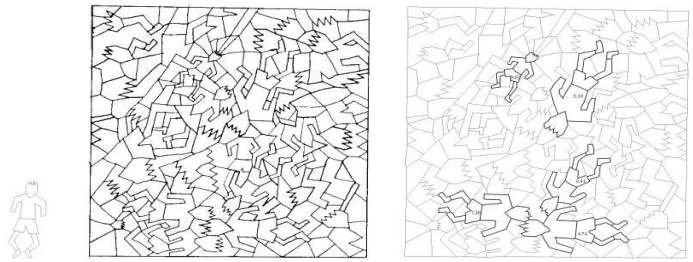
INTRODUCTION

"Premature optimization is the root of all evil".

Donald Ervin Knuth

The main objective of this thesis is modeling and analyzing solution approaches for a new optimization problem referred to as *the minimum hub cover (MHC)* problem. The problem has recently originated from a new representation used for query processing over large graph databases, which store a high volume of relational data coming from various sources including communication, social and biological networks [64, 109]. For instance, a chemical compound is a graph database where each node represents an atom and each edge represents a chemical bond formed between two atoms. For readers not familiar with graph query processing, also known as graph matching, querying a graph database refers to searching structural similarity between the nodes of a query graph and a database graph under a set of label constraints. Subgraph isomorphism problem, on the other hand, is to find whether a database graph includes a subgraph that is structurally similar to a given query graph. Formally, two graphs $G(V, E)$ and $G'(V', E')$ are *isomorphic* if there is a bijective function $f : V \rightarrow V'$ such that any two vertices u and v in G are adjacent if and only if $f(u)$ and $f(v)$ are adjacent in G'. Given two graphs G and $\bar{G}$, the *subgraph isomorphism problem* tries to find a subgraph of G that is isomorphic to $\bar{G}$. Figure 1.1 illustrates the relationship between graph query processing and subgraph isomorphism. Figure 1.1a and Figure 1.1b are the query and data graphs, respectively. These figures capture hand-drawn images of human figures.

Figure 1.1c shows all subgraphs of Figure 1.1b that are isomorphic to Figure 1.1a and thus are embedded in the data graph. The goal of subgraph matching is to identify the isomorphic graphs in Figure 1.1c given the query graph in Figure 1.1a against the data graph in Figure 1.1b. Another example can be seen in Figure 1.2, which demonstrates a query and a database graph of two molecular compounds. Note that the database graph on the right has a subgraph, which is structurally identical to the query graph on the left. Therefore, carrying out a query with this subgraph returns a positive response. The MHC problem has been recently introduced as an alternate solution method to expedite the efficiency of subgraph matching. Latest studies demonstrate that searching a graph query by solving the MHC problem improves the current techniques in graph query processing [91, 92].



(a) (b) (c)

Figure 1.1: Example for subgraph isomorphism between two hand-drawn images and the resulting solution [75]

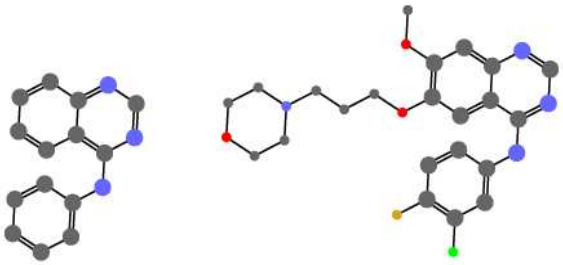


Figure 1.2: A query and a database graph of a molecular compound

The MHC problem is one of the $\mathcal{NP}$-hard problems in the literature [44]. Hence, solving the MHC problem to optimality is inherently hard. The objective of the MHC problem is to cover all edges of a graph with the minimum number of vertices. Unlike

the conventional meaning of covering, here, a selected vertex covers not only its incident edges but also the edges between its adjacent neighbors. For instance, in Figure 1.3, the edges that can be covered by vertex f are $(f,g),(f,h),(f,k)$ and (h,k) and the optimal solution is $\{a,c,f\}$. The formal definition of the MHC problem follows.

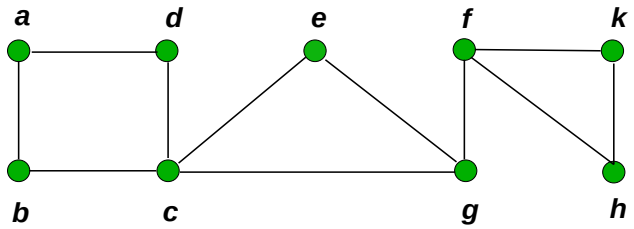


Figure 1.3: A sample graph for the minimum hub cover problem

DEFINITION 1.1 *Let $G(V,E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. Then, for a given graph G, a subset of the vertices $HC \subseteq V$ is a hub cover of G if for every edge $(i,j) \in E$, either $i \in HC$ or $j \in HC$ or there exists a vertex k such that $(i,k) \in E$ and $(j,k) \in E$ with $k \in HC$. The MHC problem is finding a hub cover that has the minimum number of vertices.*

1.1 Motivation

Many relational data coming from various sources, such as; bioinformatics, social networks, world wide web and so on, can be represented as graphs. Due to the recent explosion in graph database applications, it is very critical to manage and process the information encoded in graphs. One of the major hurdles in managing very large graph databases is to efficiently store the graphs which require large memory. The other hurdle is to develop fast graph query processing techniques to extract the information from the graphs quickly. This is important especially for those applications where high response times cannot be tolerated. MHC has been introduced to the literature as a compact graph representation model, which requires less memory. In this representation, a query graph is represented by a subset of hub nodes coming from the solution of the MHC problem [64, 109]. Also, the graph matching algorithm, which searches

subgraph isomorphism over that subset achieves reduction in both solution time and memory usage relative to other existing techniques.

Proposed graph matching algorithm in [109] is a two-phase algorithm. In the first phase, MHC is obtained for the query graph and in the second phase, a one-to-one mapping is obtained between the hub nodes in the query graph and the nodes of the database graph. In this study, we primarily focus on the first phase. From this perspective, we propose mathematical programming approaches to solve the MHC problem. Since we solve an $\mathcal{NP}$-hard problem in the first phase, it is very critical to obtain an optimal or near optimal solution to the MHC problem quickly. Our motivation is two-fold: (1) With the advances in mathematical programming techniques and computational machinery, very large $\mathcal{NP}$-hard problems, which were not solved years ago can now be solved efficiently. Therefore, we investigate the existing solution methods in mathematical programming area and adapt them for the MHC problem. (2) Since the MHC problem has been recently introduced, there is a need for exploring its various mathematical programming formulations. To this end, we give not only different mathematical programming models but also study their relaxations.

1.2 Contributions

The contributions of this thesis are as follows:

- We exploit various mathematical programming formulations of the MHC problem. We introduce new binary programming models along with a quadratic integer programming model. The relaxations of the binary models are linear programs whereas the relaxation of the last model is a semidefinite program. We also present several rounding heuristics to accompany the proposed relaxations. We conduct an extensive computational study to illustrate the empirical performances of the rounding heuristics.

- We study an approximation algorithm, which provides an approximate solution to the MHC problem on planar graphs. This algorithm can be used in various graph query processing applications, such as; biometric identification, image clas-

sification and object recognition and so on. We prove an approximation bound for the algorithm and conduct the first extensive numerical experiments to test its empirical performance [111].

- We discuss query graph processing, and the connection between the graph matching computation and MHC. We demonstrate how MHC provide strategic advantages for graph matching computations.

1.3 Outline

Chapter 2 surveys the literature related to graph query processing and the optimization problems with similar mathematical programming formulations. In Chapter 3, we present various mathematical programming formulations of MHC. The linear and semidefinite programming relaxations of MHC are also given in this chapter. Chapter 4 is dedicated to solution methods proposed for the MHC problem. We first propose solution approaches for planar graphs. We introduce an approximation algorithm, which returns approximate MHC on planar graphs with proven performance ratio. Then, we continue with the solution methods for general graphs such as greedy algorithms and relaxation heuristics. In each section, we provide computational experiments, which test the performances of the proposed solution methods. In Chapter 5, we briefly discuss graph query processing. We talk about the connection between MHC and the graph matching, and demonstrate how MHC is used in graph matching computation. A brief summary and concluding remarks as well as directions for future research are given in Chapter 6.

Chapter 2

LITERATURE REVIEW

"I don't need to know everything, I just need to know where to find it, when I need it".
Albert Einstein

This chapter surveys the relevant studies under two categories. In the first category, we summarize studies related to some well-known optimization problems, which share similar mathematical programming formulations with the MHC problem. We refer the readers to Appendix A for details about those problems. In the second category, we focus on graph query processing and survey the studies as well as its application areas.

2.1 Related Problems

The first related problem is the set covering problem (SCP), which is one of the oldest and most studied optimization problems. Many studies focus on solving SCP to optimality with exact algorithms. Exact algorithms generally rely on the branch-and-bound method to obtain optimal solutions [9, 16, 19, 41]. Beasley [16] uses subgradient optimization and a heuristic algorithm to bound the problem. Beasley and Jornsten [19] employ the same method but improve the solution quality through Gomory f-cuts with a better branching strategy. Fisher and Kedia [41] use a primal and a dual heuristic for bounding. Similarly, Balas and Carrera [9] use a primal and a dual heuristic and a dynamic subgradient procedure, and iteratively improve the bounds by variable fixing.

SCP is long known to be $\mathcal{NP}$-hard in the strong sense [44]. Similar to the problems in the same class, many algorithms have been developed to provide approximate solutions with proven performance guarantees. Gomes et al. [48], Grossman and Wool [49], Vazirani [103], and Williamson [105] list various approximation algorithms and they compare their theoretical and empirical performances. Chvatal [30] proposes a greedy-type algorithm to approximate the SCP with a performance guarantee $\log|\mathcal{S}|$. Bar et al. [12], Hall and Vohra [50], and Hochbaum [59] propose approximation algorithms using primal and dual linear programming formulations with a performance guarantee f which is defined as the maximum number of sets that can cover an item. An inapproximability result is presented by Lund and Yannakakis [77] which is a factor of $c\log|\mathcal{S}|$ for any $c < 1/4$ unless $NP \subseteq DTIME(|\mathcal{S}|^{poly\log|\mathcal{S}|})$. Bertsimas and Vohra [20] propose a randomized rounding algorithm that obtains the same performance guarantee. Bronniman and Goodrich [22] approximate SCP by using the minimum hitting set formulation when the Vapnik-Chervonenkis dimension is d [101]. Their algorithm ensures that the largest set cover is at most a factor of $O(d\log(ds))$ where s is the cardinality of optimal set cover. Then, Even et al. [38] improve the bound with a factor of 4 relative to Bronniman and Goodrich [22], i.e., the largest set cover is within at most a factor of $O(\frac{d}{4}\log(ds))$.

There are also heuristics sacrificing optimality but obtaining fairly good solutions within an acceptable time without performance guarantees. Caprara et al. [25], Gomes et al. [48], and Grossman and Wool [49] list various heuristics and approximation algorithms and show that those algorithms perform well empirically. There are several approaches to develop a heuristic algorithm. Among these, we have greedy algorithms, randomized search, heuristics based on linear programming and Lagrangian relaxations, and the closely related, primal-dual methods. The simplest algorithms are the greedy algorithms, which can be used to solve large-scale set covering problems in negligible times. However, their myopic nature may easily yield solutions far from optimality. Haouari and Chaouachi [55], Feo and Resende [39], as well as Vasko and Wilson [102] introduce randomness and penalization into the greedy algorithms to improve solution quality. Along this line, three local search heuristics can be mentioned [71, 78, 106]. Fin-

ger et al. [40] conduct an analysis on benchmark instances by measuring the correlation between the cost of a solution and the closeness to the optimal solution. This study gives useful insights to understand the problem structure and develop problem-specific local search algorithms. Several meta-heuristics have also been proposed for SCP. Among these, we can list simulated annealing [23, 62], genetic algorithms [2, 18, 76], tabu search [27, 68, 81], ant colony optimization [89], and electromagnetism meta-heuristic [6]. In a recent study, Muter et al. [82] devise a generic framework that uses information from the linear programming relaxation for promoting meta-heuristics to diversify or intensify while searching for the optimum of set covering-type optimization problems. Muter et al. [82] also consider the role of dual information in their numerical study on the vehicle routing problem with time windows. First, they use the dual information for altering the randomized selection mechanism in the meta-heuristic. With this new mechanism, the meta-heuristic is encouraged to generate routes (sets) that are more likely to have negative reduced costs. Second, the dual information is used to reduce the size of the column pool by removing those columns with higher reduced costs. Muter et al. [82] report that the dual information does not increase the effectiveness of their algorithms. However, Yelbay et al. [110] assert the contrary through a fundamentally different setting and implementation. Yelbay et al. [110] emphasize the importance of using the optimal dual solution of the linear programming relaxation of SCP. The dual information is gathered from the optimal solution of the linear programming relaxation, and then problem size is reduced considerably so that the resulting SCP can be solved by an integer programming solver with much less computational effort.

Moreover, several studies design heuristics based on the Lagrangian relaxation or the linear programming relaxation of SCP [17, 24, 28, 59, 100]. The resulting primal-dual approach has been commonly used for approximating $\mathcal{NP}$-hard optimization problems that can be modeled as integer programming problems, such as the metric traveling salesman problem, the Steiner tree problem, the Steiner network problem, and the set covering problem [103]. Bar and Even [11] are the first researchers who have considered a generic primal-dual approach to approximate the set covering problem – later shown to be equivalent to the local ratio technique [13]. The basis of the primal-dual approach

is finding only a feasible solution to the dual of the linear programming relaxation of the SCP. Using this solution, an integral solution for SCP is constructed. Although the worst case performance of the primal-dual algorithm of Bar and Even [11] is poor [50], its empirical performance turns out to be much more promising. Therefore, several studies have sprung out of the primal-dual approach in the set covering literature [20, 79, 105, 108].

In the literature, there are many studies that use the dual information from the Lagrangian relaxation of SCP to reduce the size of the large-scale SCP instances by variable fixing. Reduced costs are computed for a current set of Lagrangian multipliers attained by a subgradient procedure. Beasley [17], Caprara et al. [24], Ceria et al. [28], and Yagiura et al. [106] set a variable to zero, whenever its reduced cost is greater than a threshold. The dual information is also used to construct a good feasible solution. The variables with the most negative reduced costs are accepted as good candidates to obtain a feasible solution [24, 27].

The second closely related problem is minimum vertex cover (MVC), which is a special case of the unicost SCP. MVC is equivalent to MHC in triangle-free graphs. In the literature, there are various approximation algorithms for MVC. However, it is known that approximating MVC with a good performance ratio is also difficult. Dinur and Safra [34] show that finding a δ-approximate solution for the MVC problem in polynomial time is $\mathcal{NP}$-hard for $\delta \leq 1.3606$. Moreover, Khot and Regev [67] show that it is $\mathcal{NP}$-hard to find an approximate solution within a constant factor less than 2. Nagamochi and Ibaraki [83] propose an odd-cycle elimination technique and achieve a bound $2 - \frac{8m}{(13n^2+8m)}$. Hochbaum [59] proposes a 2-approximation algorithm, which uses the linear programming relaxation of MVC to approximate the optimal integer programming solution. Arora et al. [3] add odd-cycle inequalities to obtain a tighter bound, $2 - o(1)$. Similarly, Han et al. [53] use the solution of this new formulation and apply some new graph reduction rules to approximate the optimal vertex cover with a performance ratio 3/2 for special types of graphs. Asgeirsson and Stein [4, 5] propose an algorithm that performs a set of reduction rules and guarantees 3/2 performance ratio. However, their algorithm does not always guarantee a feasible solution. Han and

Punnen [52, 54] obtain a $2-[1/(1+\sigma)]$ approximation ratio where σ is an upper bound on a measure related to weak edge of a graph which is defined as an edge with only one end point in the optimal vertex cover. Dharwadker [33] proposes an algorithm which performs a series of redundant vertex elimination operations and gives an approximation ratio $n-[\frac{n}{1+\Delta}]$ where Δ is the maximum degree of a vertex. Some of the researchers focus on approximating MVC for the more generalized hypergraphs in which an edge can connect any number of vertices. Okun [87] achieves an approximation bound $(D-1)[1-\Delta^{1/(1-D)}]+1$, where the number of neighbors are bounded by D. Similarly, Cardinal et al. [26] propose an approximation algorithm for dense hypergraphs with bounded size k. Their algorithm guarantees an approximation bound $\frac{k}{1+(k-1)\frac{d}{k\Delta}}$ where d is the average degree of a vertex. Semidefinite programming relaxations of the MVC problem can also be used to achieve a good performance ratio. Halperin [51] gives a factor of $2-(1-o(1))\frac{2\ln\ln\Delta}{\ln\Delta}$ for the MVC problem. He also uses the similar techniques to approximate k-uniform hypergraphs with a ratio of $k-(1-o(1))\frac{k(k-1)\ln\ln\Delta}{ln\Delta}$ for large n. Similarly, Karakostas [66] adds triangle inequalities to the semidefinite programming formulation of the MVC problem and obtains a tighter bound given as $2-\Theta(\frac{1}{\sqrt{\log n}})$.

2.2 Graph Query Processing

Studies related to query processing can be categorized into two groups: exact and inexact graph matching algorithms. Exact graph matching algorithms are proposed to solve either the graph isomorphism problem [32, 93] or the more general subgraph isomorphism problem [32, 65, 74, 97, 99, 104, 109, 114].

A survey of recent graph pattern matching algorithms can be found in [43, 72]. Among those studies, we have studies applying index-based searching [97, 104]. Shang [97] propose a two-phase graph matching algorithm. In the first phase, indexing helps to decrease the number of candidates mappings. In the second phase, subgraph isomorphism computations are performed among the candidates. Similarly, Weber et al. [104] propose an index-based graph matching algorithm. Permutations of vertex sequences in an adjacency matrix is represented by a tree. Firstly, they assign each tree a label and then a weight so that they can eliminate some vertex sequences from consideration.

In this way, the number of permutations for searching decreases considerably. Lipets et al. [74] propose graph decomposition methods and subgraph isomorphism search is performed individually for each subgraph separately. Moreover, Jamil [64] and Zhu et al. [114] propose using new data structures and graph representations keeping topological data. Zhu et al. [114] and Jamil [64] and Yelbay et al. [109] propose vector signature and graphlet representation, which keep topological data to help the detect the most similar vertices between the nodes of query and database graphs. Graphlet representation uses the solution of the MHC problem to represent the data as a compact form. Rivero and Jamil [91, 92], and Yelbay et al. [109] propose an algorithm using the graphlet representation and searching over the hub nodes obtained by solving the MHC problem. Rivero and Jamil [91, 92] show that solving the MHC problem increases the efficiency of the graph query processing relative to its counterpart algorithms. Some studies especially focus on labelled graph databases [56, 97, 112, 113]. These systems pay particular attention to graph specific properties to increase the efficiency of graph query processing and thus they each favor specific graph types, e.g., arbitrary, tree or path type queries and graphs.

Some of the studies are specialized for planar graph databases constructed from images [31]. Since hand-written or digital images may include noise, inexact graph and/or subgraph isomorphism are used to respond to a query of databases constructed from images. Images are first partitioned into regions. Each region is represented by a vertex and two adjacent regions are linked by an edge. Dorn [36], Eppstein [37], Kukluk [70], and Messmer and Bunke [80] apply planar graph decomposition techniques to find subgraph and/or graph isomorphism. Eppstein [37] propose a polynomial algorithm to solve the subgraph isomorphism in planar graphs. They use Baker's decomposition technique in [7] to partition a planar graph into a set of subgraphs. Each subgraph is then solved by dynamic programming algorithm proposed in [7]. Jaja and Kosaraju [63] and Gazit and Reif [46] use parallel processing for computing graph isomorphism. Lingas [73] and Higuera et al. [58] focus on subgraph isomorphism for special types of planar graphs, open graphs in which some faces are invisible. Open plane graphs are generally used to represent an image that represents a robot moves in such a way that

the forbidden areas are labelled as invisible. Neuhaus et al. [86], Saux and Bunke [95], and Yates and Valiente [107] propose inexact graph matching algorithms to identify not the exact but the most similar graphs of images. Graph edit distance is computed as a similarity measure. Abdulrahim and Misra [1], Baloch and Krim [10], and Saxena et al. [96] develop graph and/or subgraph isomorphism algorithms for object recognition, identifying an object by comparing it to a database of known objects. Moreover, some studies focus on biometric identification of digital images such as faces, hand postures and fingerprints. Chikkerur et al. [29], Isenor and Zaky [61], and Neuhaus and Bunke [84, 85] propose several graph representation methods and graph matching algorithms to identify a match in the fingerprint database. After filtering and removing the noise in the images, each bifurcation point is represented by a vertex and vertices are connected by an edge if there is a ridge between two two bifurcation points. Similar to other image applications, graph edit distance is computed to find the set of best matches. Moreover, Llados et al. [75] propose an algorithm for symbol recognition to measure the similarity between two hand-written documents.

Chapter 3

MATHEMATICAL MODELING

"The formulation of the problem is often more essential than its solution, which may be merely a matter of mathematical or experimental skill."

Albert Einstein

In this chapter, we focus on mathematical programming models and their relaxations of the MHC problem. We give two binary integer programming (IP) formulations as well as a novel quadratic integer programming formulation. We present the linear programming relaxations of the first two binary integer programming models. Then, we introduce a semidefinite programming relaxation obtained from the quadratic integer programming model.

3.1 Mathematical Programming Formulations

In this section, we first prove that the MHC problem is one of the $\mathcal{NP}$-hard problems in the literature and also decision version of it, $(\mathcal{MHC}-\mathcal{D})$, belongs to the class of $\mathcal{NP}$-complete problems [44]. Then, we introduce mathematical programming formulations and their relaxations. The definition of $(\mathcal{MHC}-\mathcal{D})$ and its proof is given as follows:

DEFINITION 3.1 *Given a graph $G(V,E)$ and an integer $n \leq |V|$, does there exist a subset $HC \subseteq V$ with $|V| \leq n$ such that for every edge $(i,j) \in E$, $i \in HC$ or $j \in HC$ or there exists a vertex $k \in HC$ such that both $(i,k) \in E$ and $(j,k) \in E$.*

THEOREM 3.1 *The $\mathcal{MHC}-\mathcal{D}$ problem is $\mathcal{NP}$-complete.*

PROOF. Given a yes-instance and $HC' \subseteq V$ with $|HC'| \leq n$, we can verify in polynomial time that every edge in E is covered by HC'. Thus, the $\mathcal{MHC}-\mathcal{D}$ problem is in $\mathcal{NP}$. Poljak [88] proves that the MVC problem is $\mathcal{NP}$-complete for triangle-free graphs. Therefore, we now complete the proof by a reduction from the MVC problem on triangle-free graphs. In triangle-free graphs, for all $(i,j) \in E$ there does not exist a vertex k such that $(i,k) \in E$ and $(j,k) \in E$. Thus, either i or j must be in HC' to cover the edge (i,j). Consequently, the MHC and the MVC of a graph are equivalent in this class of graphs. □

Our first formulation is a set covering formulation of MHC introduced in [109]. If an edge corresponds to an item, and a set is defined for each vertex whose elements are the edges covered by that vertex, then the connection between the set covering and MHC can easily be established. This mathematical programming formulation is given as follows:

$$\text{minimize} \quad \sum_{j \in V} x_j, \tag{3.1}$$

$$\text{subject to} \quad x_i + x_j + \sum_{k \in \mathcal{K}^{(i,j)}} x_k \geq 1, \qquad (i,j) \in E, \tag{3.2}$$

$$x_j \in \{0,1\}, \qquad j \in V. \tag{3.3}$$

Here, x_j is a binary variable, which is equal to 1 when vertex j is selected. For $(i,j) \in E$, $\mathcal{K}^{(i,j)}$ denotes all those vertices $k \in V$ such that $(i,k) \in E$ and $(j,k) \in E$. The objective function (3.1) minimizes the number of selected vertices. Constraints (3.2) ensure that every edge is covered by at least one vertex in the hub cover. Finally, constraints (3.3) enforce binary restrictions on the variables.

The well-known MVC problem is a special case of the MHC problem when the cardinality of the set $\mathcal{K}^{(i,j)}$ is zero; that is, $|\mathcal{K}^{(i,j)}| = 0$. The MVC problem has a complementary problem formulation known as the maximum independent set problem. This relationship inspired us to introduce a new optimization problem, which we call

the maximum triangular set (MTS) problem. The formal definition of MTS follows.

DEFINITION 3.2 *For a given graph $G = (V, E)$, $TS \subseteq V$ is a triangular set if and only if for every edge (i, j) at most $|\mathcal{K}^{(i,j)}| + 1$ of the vertices in $\bar{\mathcal{K}}^{(i,j)} := \mathcal{K}^{(i,j)} \cup \{i, j\}$ are also in TS. The MTS problem is about finding a triangular set which has the maximum number of vertices.*

A careful reader may notice that MTS is equivalent to MHC in the sense that the solution of one problem will yield a solution for the other one. This is in fact the case as we prove in Lemma 3.1.

LEMMA 3.1 *In any graph $G = (V, E)$, HC is a hub cover in G if and only if $V \setminus HC$ is a triangular set or TS is a triangular set in G if and only if $V \setminus TS$ is a hub cover.*

PROOF. Suppose TS is a triangular set on G. Then for any edge (i, j), at most $|\mathcal{K}^{(i,j)}|+1$ of the vertices in $\bar{\mathcal{K}}^{(i,j)}$ are in TS. Since the number of vertices that can cover edge (i, j) is equal to $|\mathcal{K}^{(i,j)}| + 2$, at least one of the vertices in S must be in $V \setminus TS$. Thus, $V \setminus TS$ must be a hub cover. Conversely, suppose $V \setminus TS$ is a hub cover. Then, at least one of the vertices in $\bar{\mathcal{K}}^{(i,j)}$ must be in $V \setminus TS$ so that $V \setminus TS$ is a hub cover. That is for each edge, the cardinality of the subset of $\bar{\mathcal{K}}^{(i,j)}$ included in TS is less than $|\mathcal{K}^{(i,j)}| + 2$ and hence, TS is a triangular set. □

The mathematical programming model of the MTS problem is given by

$$\text{maximize} \quad \sum_{j \in V} x_j, \tag{3.4}$$

$$\text{subject to} \quad x_i + x_j + \sum_{k \in \mathcal{K}^{(i,j)}} x_k \leq |\mathcal{K}^{(i,j)}| + 1, \qquad (i, j) \in E, \tag{3.5}$$

$$x_j \in \{0, 1\}, \qquad j \in V, \tag{3.6}$$

where x_j is a binary variable that is equal to 1 when vertex j is selected. The objective function (3.4) maximizes the number of selected vertices. Constraints (3.5) ensure that the solution is a TS as defined in Definition 3.2. The final set of constraints (3.6) ensure the integrality of the binary variables.

Our last reformulation is a quadratic integer program, where each term is a product of two binary variables. This formulation shall form the basis of the semidefinite programming relaxation that we will introduce in Section 3.2:

$$\text{minimize} \quad \sum_{j \in V} (1 + y_0 y_j)/2, \tag{3.7}$$

$$\text{subject to } (y_0 - y_i)(y_0 - y_j) + (2y_0 - y_i - y_j) \sum_{k \in \mathcal{K}^{(i,j)}} (y_0 - y_k) \leq 8|\mathcal{K}^{(i,j)}|, \qquad (i,j) \in E, \tag{3.8}$$

$$y_j \in \{+1, -1\}, \qquad j \in V \cup \{0\}. \tag{3.9}$$

The optimal solution of the MHC problem is given by those vertices $j \in V$ such that $y_j = y_0$. The set of constraints (3.8) is obtained after simplifying the following constraint for each $(i,j) \in E$.

$$(y_0 - y_i)(y_0 - y_j) + \sum_{k \in \mathcal{K}^{(i,j)}} (y_0 - y_i)(y_0 - y_k) + \sum_{k \in \mathcal{K}^{(i,j)}} (y_0 - y_j)(y_0 - y_k) \leq 8|\mathcal{K}^{(i,j)}|. \tag{3.10}$$

The following example illustrates relation (3.10) on a clique of three vertices.

EXAMPLE 3.1 *Suppose that we consider the clique consisting of the vertices i, j, and k. Since in a clique, every two vertices are connected by an edge, the constraint* (3.10) *for edge (i,j) becomes*

$$(y_0 - y_i)(y_0 - y_j) + (y_0 - y_i)(y_0 - y_k) + (y_0 - y_j)(y_0 - y_k) \leq 8. \tag{3.11}$$

Given $y_i \in \{-1, +1\}$, this constraint ensures that the solution $y_0 \neq y_i = y_j = y_k$ is infeasible. Thus, at least one of the three vertices is selected.

Clearly, the mathematical programming models that we have introduced in this section are very difficult to solve to optimality. Nonetheless, the relaxations of these problems can be solved efficiently. These relaxations have two uses: (i) Their optimal

objective function values can be used to give bounds. Then, these bounds could be used to increase the efficiency of exact methods. (ii) The optimal solutions of the relaxations can be used to obtain feasible solutions for the original problem. In certain cases, these solutions can even play a role to give approximation bounds. These relaxations are given in the next section.

3.2 Mathematical Programming Relaxations

The mathematical programming relaxation is a modeling approach to replace a difficult problem with easier one. In most cases, the solutions of the relaxed models are not feasible for the original problem. Then, one needs to resort to rounding heuristics. In this section, we focus on linear and semidefinite programming relaxations. In next chapter, we present several rounding heuristics using those relaxations.

3.2.1 Linear Programming Relaxation

The linear programming (LP) relaxation is obtained simply by replacing the binary constraints on the variables with the inequalities $0 \leq x_j \leq 1$. We relax the integrality constraint in models (3.1)-(3.3) and (3.4)-(3.6) and obtain LP models for the MHC and the MTS problems that we shall refer to as LP1 and LP2, respectively.

Associated with LP1, there is a corresponding dual formulation given is as follows:

$$\text{maximize} \quad \sum_{(i,j)\in E} y_{(i,j)}, \tag{3.12}$$

$$\text{subject to} \quad \sum_{(i,j)\in E} y_{(i,j)} + \sum_{(j,i)\in E} y_{(j,i)} + \sum_{j\in\mathcal{K}^{(i,k)}} y_{(i,k)} \leq 1, \qquad j \in V, \tag{3.13}$$

$$y_{(i,j)} \geq 0, \qquad (i,j) \in E, \tag{3.14}$$

where $y_{(i,j)}$ is a dual variable corresponding to the coverage constraint for edge (i,j).

3.2.2 Semidefinite Programming Relaxation

Semidefinite programming (SDP) is about optimizing a linear function of a symmetric matrix over the cone of positive semidefinite matrices. LP is a special case of SDP. Today, many $\mathcal{NP}$-hard optimization problems have semidefinite relaxations. The important point is that very good approximation bounds can be obtained after solving the SDP relaxations of hard combinatorial problems [47, 51, 66].

Before introducing the SDP relaxation, we first remove the integrality constraint from (3.7)-(3.9) and obtain

$$\text{minimize} \quad \sum_{j\in V}(1+y_0y_j)/2, \tag{3.15}$$

$$\text{subject to } (y_0-y_i)(y_0-y_j)+(2y_0-y_i-y_j)\sum_{k\in\mathcal{K}^{(i,j)}}(y_0-y_k)\le 8|\mathcal{K}^{(i,j)}|, \qquad (i,j)\in E, \tag{3.16}$$

$$y_j^2=1, \qquad j\in V\cup\{0\}. \tag{3.17}$$

We next introduce the matrix variable $\mathbf{Y}=\mathbf{y}\mathbf{y}^T$, where $\mathbf{y}$ is the vector consisting of components y_0 and $y_i, i\in V$. We also define $\mathbf{A}\bullet\mathbf{B}$:=trace$(\mathbf{A}^T\mathbf{B})$. Using now this notation, we can give the following equivalent formulation:

$$\text{minimize} \quad \mathbf{C}\bullet\mathbf{Y} \tag{3.18}$$

$$\text{subject to} \quad \mathbf{A}^{(i,j)}\bullet\mathbf{Y}\le 8|\mathcal{K}^{(i,j)}|, \qquad (i,j)\in E, \tag{3.19}$$

$$\text{diag}(\mathbf{Y})=\mathbf{e}, \tag{3.20}$$

$$\mathbf{Y}\succeq 0, \tag{3.21}$$

$$\text{rank}(\mathbf{Y})=1, \tag{3.22}$$

where $\mathbf{C}$ and $\mathbf{A}^{(i,j)}$ are symmetric matrices, $\mathbf{e}$ is the vector of ones and $\mathbf{Y}\succeq 0$ means that the matrix $\mathbf{Y}$ is positive definite. Before specifying $\mathbf{C}$ and $\mathbf{A}^{(i,j)}$, let us relax the

constraint (3.22) and obtain the SDP relaxation of the MHC problem given by

$$\begin{aligned}
\text{minimize} \quad & \mathbf{C} \bullet \mathbf{Y} && && (3.23)\\
\text{subject to} \quad & \mathbf{A}^{(i,j)} \bullet \mathbf{Y} \leq 8|\mathcal{K}^{(i,j)}|, && (i,j) \in E, && (3.24)\\
& \text{diag}(\mathbf{Y}) = \mathbf{e}, && && (3.25)\\
& \mathbf{Y} \succeq 0. && && (3.26)
\end{aligned}$$

The symmetric matrices in the SDP relaxation are defined as follows: Let C_{mn} denote the components of the matrix $\mathbf{C}$. Then,

$$C_{mn} = \begin{cases} 1/4, & \text{if } m = 0 \text{ and } n \in V; \\ 1/4, & \text{if } m \in V \text{ and } n = 0; \\ 0, & \text{otherwise.} \end{cases}$$

When it comes to the matrix $\mathbf{A}^{(i,j)}$, we observe that

$$(y_0 - y_i)(y_0 - y_j) = \mathbf{M} \bullet \mathbf{y}\mathbf{y}^T,$$

where $\mathbf{M}$ is a symmetric matrix and its nonzero components are given by

$$M_{00} = 1, \quad M_{0i} = M_{i0} = M_{0j} = M_{j0} = -1/2, \quad M_{ij} = M_{ji} = 1/2.$$

Note for a given $(i,j) \in E$ that the constraint (3.10) is constructed by summing up matrices like $\mathbf{M}$ above. Consequently, the matrix $\mathbf{A}^{(i,j)}$ also becomes a symmetric matrix.

Formally, we write $\mathbf{Y} = \mathbf{V}^T\mathbf{V}$, where the columns of $\mathbf{V}$ are given by $\mathbf{v}_m, m \in$

$V \cup \{0\}$. Then, we obtain,

$$\text{minimize} \quad \sum_{m,n} C_{mn} \mathbf{v}_m^T \mathbf{v}_n \tag{3.27}$$

$$\text{subject to} \quad \sum_{m,n} \mathbf{A}_{mn}^{(i,j)} \mathbf{v}_m^T \mathbf{v}_n \leq 8|\mathcal{K}^{(i,j)}|, \qquad (i,j) \in E, \tag{3.28}$$

$$\mathbf{v}_m^T \mathbf{v}_n = 1, \qquad m \in V \cup \{0\}, \tag{3.29}$$

$$\mathbf{v}_m \in \mathbb{R}^{|V|+1}, \qquad m \in V \cup \{0\}. \tag{3.30}$$

Chapter 4

SOLUTION METHODS

"If I had 60 minutes to solve a problem, I'd spend 55 minutes defining it and 5 minutes solving it".

Albert Einstein

This chapter provides an overview of the solution methods for the MHC problem. The first chapter summarizes the solution methods proposed for the planar graphs which have many applications in graph query processing. The second chapter is about the algorithms proposed for the problems that share similar mathematical programming formulations with the MHC problem. We present some computational results to demonstrate the performance of those solution methods on the MHC problem. Third section introduces some rounding algorithms to solve the MHC problem on general graphs.

4.1 Planar Graphs

In this section, we focus on planar graphs for they appear in various graph query processing applications. A *planar graph* is a graph that can be embedded in a plane in such a way that no edges cross each other. Many graph databases in query processing satisfy planarity condition that is common in diverse applications, such as; face recognition, fingerprint identification, hand posture recognition, image classification, object recognition, and so on. As an example, Figure 4.1 shows the planar graph representation of a finger print. Each node in the graph represents a finger ridge pattern and the edges

are constructed according to the orientation of the ridges.

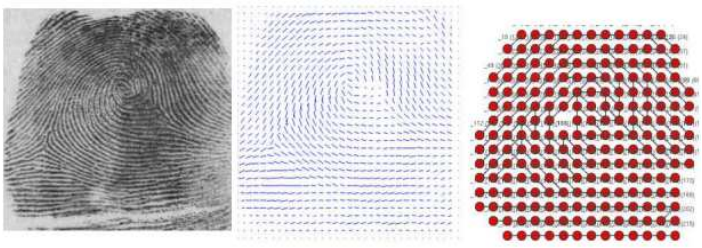


Figure 4.1: Graph representation of a fingerprint image [85]

MHC on planar graphs is $\mathcal{NP}$-complete based on the fact that MVC is $\mathcal{NP}$-complete when restricted to triangle-free planar graphs [45]. In Chapter 3, we use the equivalence of MHC and MVC in triangle-free graphs to prove the $\mathcal{NP}$-completeness of MHC for general graphs. We use the same argument to prove the the following corollary.

COROLLARY 4.1 *The MHC problem on planar graphs is $\mathcal{NP}$-complete.*

PROOF. Given a yes-instance and $HC' \subseteq V$ with $|HC'| \leq n$, we can verify in polynomial time that every edge in E is covered by HC'. Thus, the problem is in $\mathcal{NP}$. Minimum vertex cover problem is $\mathcal{NP}$-complete when restricted to triangle-free planar graphs [45]. In triangle-free planar graphs, for all $(i, j) \in E$ there does not exist a vertex k such that $(i, k) \in E$ and $(j, k) \in E$. Thus, either i or j must be in HC' to cover the edge (i, j). Consequently, the MHC and the MVC of a planar graph are equivalent in this class of graphs. □

Similar to other problems that belong to the class of $\mathcal{NP}$-complete problems, unless $\mathcal{NP} = \mathcal{P}$, there are hard MHC instances that are intractable with the exact methods. Therefore, we need to develop approximation algorithms with a performance guarantee or heuristics to solve the MHC problem efficiently. In line with this purpose, we discuss a well-known graph decomposition technique that partitions the graph into a set of outerplanar graphs [7]. First, we introduce an algorithm which provides an approximate solution with a proven performance ratio. We conduct a comprehensive computational experiment to investigate the empirical performance of the algorithm. Computational results demonstrate that the empirical performance of the algorithm

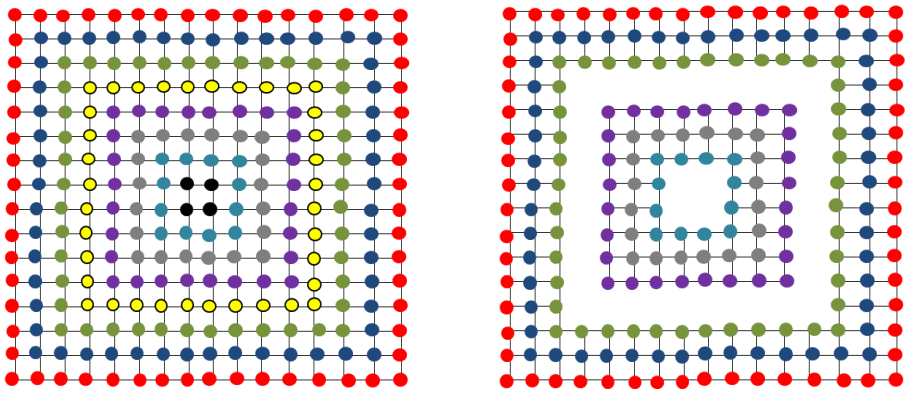
surpasses its guaranteed performance. We also apply the same decomposition approach to develop a decomposition-based heuristic, which is much more efficient than the approximation algorithm in terms of computation time. Computational results also indicate that the efficacy of the decomposition-based heuristic in terms of solution quality is comparable to that of the approximation algorithm. Finally, we discuss a dynamic programming (DP) algorithm to solve the MHC problem on outerplanar graphs to optimality.

4.1.1 Approximation Algorithm

In this section, we make use of a general decomposition technique first proposed by Baker [7]. Before discussing the technique, let us introduce some terminology used throughout this section.

Some Definitions and Terminology: A *face* of a planar graph is a region bounded by edges. A vertex of a planar graph is at *level* 1 if it is on the exterior face. A planar embedding is *k-level* if it has no nodes of level greater than k. A graph is *outerplanar* if it is a planar such that all of the vertices belong to the exterior face. A planar embedding is said to be *k-outerplanar* if removing the vertices on the exterior face results in a $(k-1)$-outerplanar embedding. Every planar graph is k-outer planar for some k. See Figure 4.2 for induced subgraph and planar embedding representations.

The graph decomposition technique can be applied to any planar graph whose planar embedding and the set of vertices in its each level are known. In case they are not known, one of the algorithms in the literature can be applied to obtain a planar embedding [21, 60]. With the proposed technique, given a planar embedding and a nonnegative number k, the planar graph is decomposed into a set of overlapping $(k+1)$-outerplanar graphs such that the union of the optimal solutions of those graphs gives a feasible solution to the original planar graph. The algorithm picks the best of these solutions as its approximation to the optimal hub cover. Figure 4.3 and Figure 4.4 illustrate how the decomposition is applied to the problem shown in Figure 4.2a when $k = 2$. The unions of the optimal solutions of the subgraphs in Figure 4.3 and

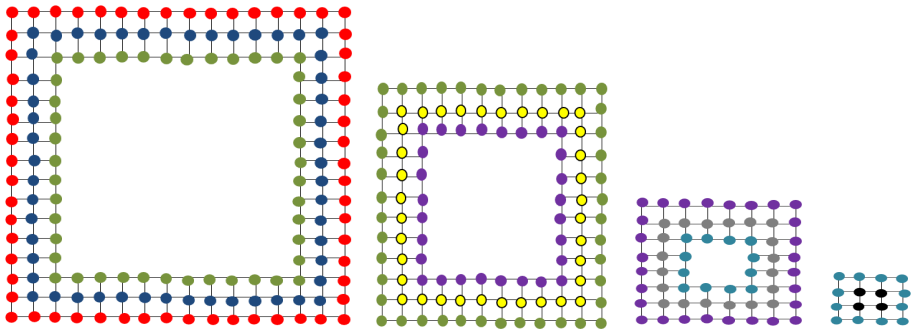


(a) The set of vertices in each level of a planar graph. The vertices of each level are represented by a different color.

(b) A subgraph which is the union of the subgraphs of the planar graph in Figure 4.2a induced by levels 1,2,3 and 5,6,7

Figure 4.2: An 8-level planar graph embedding

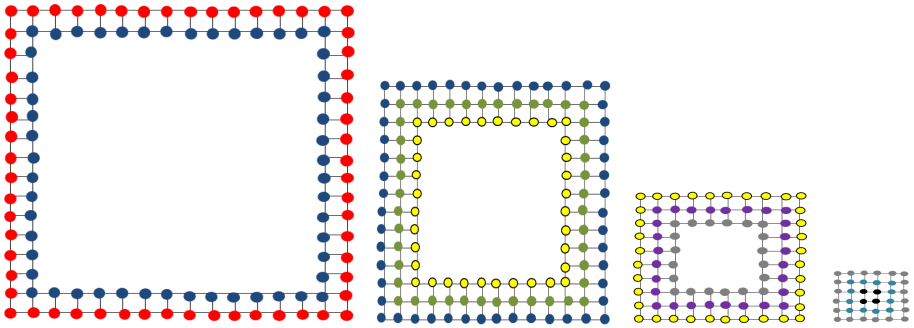
Figure 4.4 provide two different hub covers. The algorithm selects the solution with minimum cardinality as an approximate solution. Optimal solutions may be obtained by solving the IP formulation (3.1)-(3.3) of each subproblem by using an off-the-shelf solver like `CPLEX`.



(a) 1-3 (b) 3-5 (c) 5-7 (d) 7-8

Figure 4.3: The overlapping 3-outerplanar graphs when $i = 1$ and $k = 2$

The steps of the proposed decomposition and the solution approach are detailed in Algorithm 4.1. The algorithm takes a planar embedding of a graph and the decomposition parameter k as input and returns an approximate hub cover HC^{approx}. Let S_j^i be a $(k+1)$-outerplanar graph induced by levels $jk+i$ to $(j+1)k+i$ and S_j^{*i} is the optimal solution of S_j^i. A graph G' is an induced graph of G if G' is isomorphic to a graph whose vertex set V' is a subset of the vertex set V of G, and whose edge set E' consists of all those edges of G with both end vertices in V'. For instance, Figures



(a) 1-2 (b) 2-4 (c) 4-6 (d) 6-8

Figure 4.4: The overlapping 3-outerplanar graphs when $i = 2$ and $k = 2$

4.3a-4.3d demonstrate $S_0^1, S_1^1, S_2^1, S_3^1$ for $i = 1$. In lines 5 to 13, for each partition, Algorithm 4.1 iterates as follows: In line 8, a subgraph lying between boundary levels $jk + i$ and $(j + 1)k + i$ is obtained. Then, in lines 9 and 10, the IP formulation of that subgraph is solved and the solution is added to current partial feasible solution of partition i denoted as HC^i. When the algorithm exits the inner loop, a feasible MHC is obtained for the partition i. After iterating for all partitions, in line 14, the solution with minimum cardinality is selected as an approximate hub cover.

Algorithm 4.1 Planar Graph Approximation Algorithm

1: **Input:** A planar embedding of G, the vertices lying in each level, and k
2: **Output:** HC^{approx}
3: $HC^i \leftarrow \emptyset$ for each $i \in \{1, 2, \cdots, k\}$
4: $HC^{approx} \leftarrow \emptyset$
5: **for** $i := 1$ **to** k **do**
6: $\quad j \leftarrow 0$
7: $\quad$ **while** $(j + 1)k + i$th level of G is available **do**
8: $\quad\quad$ Obtain the subgraph, S_j^i, induced by levels $jk + i$ to $(j + 1)k + i$
9: $\quad\quad$ Solve (3.1)-(3.3) for the subgraph S_j^i and obtain the optimal solution, S_j^{*i}
10: $\quad\quad$ $HC^i = HC^i \cup S_j^{*i}$
11: $\quad\quad$ $j \leftarrow j + 1$
12: $\quad$ **end while**
13: **end for**
14: **Return** $HC^{approx} \leftarrow HC^p$, where $HC^p = \arg\min\{|HC^i| \ | 1 \leq i \leq k\}$

The decomposition technique guarantees a feasible solution which is within a factor of $(k + 1)/k$ from the optimal solution for a given k, where $k \geq 1$. Proposition 4.1 gives a formal proof of this statement.

PROPOSITION 4.1 *Algorithm 4.1 finds an approximate hub cover for a planar graph which is at most* $(k+1)/k$ *optimal.*

PROOF. With the decomposition approach, the boundary levels of $(k+1)$-outerplanar graphs, i.e. the overlapping levels, partition the graph into k pieces. Let V_i be the set of all vertices in the overlapping levels for each i, $1 \leq i \leq k$. In Figure 4.2a, V_1 and V_2 are the vertices lying in levels 1, 3, 5, 7 and 2, 4, 6, 8, respectively, when $k = 2$. Since the decomposition partitions the graph into k pieces, there exists at least one partition i such that at most $1/k$ of the vertices in HC^{opt} are included in V_i, where HC^{opt} is the optimal MHC in G. For each i, the union over j of the solutions gives a hub cover for the whole graph. Since only the vertices in V_i are counted twice, the cardinality of the solution is at most as follows:

$$|HC^{opt}| \leq \bigcup_j S_j^{*i} \leq |HC^{opt}| + |HC^{opt}|/k \leq (k+1)|HC^{opt}|/k. \tag{4.1}$$

This completes the proof. □

Computational Considerations. Notice that the decomposition technique splits the problem into a set of subproblems that are independent from each other. This structure of the algorithm enables us to use a parallel implementation to solve the subproblems concurrently. Such an implementation not only saves a considerable amount of computation time but it also allows handling extremely large problems for which even storing the graph in computer memory is a big burden.

Algorithm 4.1 generates feasible solutions that are obtained by taking the union of the optimal solutions of the subproblems. We observe that, if the subproblems have alternate optimal solutions, then the cardinality of the feasible solution may not be unique. Depending on the alternate optimal solution selected for each subgraph, the union, that is the cardinality of the solution, may change. Therefore, we added a subroutine to decrease the cardinality of the solution by decreasing the double coverages in the levels between two neighboring subproblems. The subroutine checks the optimal solution of the subproblem j and then perturbs the objective function coefficients of the

neighboring subproblem $(j+1)$ before solving it. The objective function coefficients of the variables that are optimal in the jth subproblem are set to $1-\epsilon$ in the $(j+1)$th subproblem where ϵ is a small non-negative number between 0 and 1. The subroutine helps neighboring subproblems generate similar optimal solutions, if there exists such an optimal solution.

Next comes the fast heuristic that we mentioned at the beginning of this chapter. The computation time of the approximation algorithm increases with k. As an alternate approach, we propose a *decomposition-based heuristic* which selects a partition i randomly among k different partitions. Then, we solve the subproblems resulting from partition i and take the union of the optimal solutions of those subproblems. The decomposition-based heuristic does not guarantee a performance ratio but it provides a feasible solution whose computation time is $1/k$ of that of the approximation algorithm.

Numerical Experiments. Approximation algorithms provide solutions with proven performance guarantees for computationally intractable problems. However, the bounds suggested by the theory are usually quite conservative. In this section, we conduct a set of experiments to compare the theoretical bound $(1+1/k)$ against the empirical performance of Algorithm 4.1. We also test how well the decomposition-based heuristic performs.

Before delving into the details, let us define the instances and the experimental setup. The proposed approximation algorithm and the decomposition-based heuristic were tested on synthetically generated planar graphs with known planar embeddings. Our problem set includes 20 planar graphs with different sizes from small to large. The numbers of vertices and edges range from several thousands to a million. The number of levels, on the other hand, ranges from 100 to 5,000. Optimal IP solutions were obtained by `IBM ILOG CPLEX Optimization Studio 12.6` running on a personal computer with Intel Xeon CPU E5-2630 and 64 GB of RAM. The upper limit on the solution time is set to 3,600 seconds for the `CPLEX` solver. The batch processing of the instances is carried out through C++ scripts. We used C++ libraries named Boost Asio and Thread to execute the algorithm in parallel.

Figure 4.5 shows how the empirical and theoretical performances of the approximation algorithm and the decomposition-based heuristic change with k. The theoretical performance of the approximation algorithm improves with increasing k and the optimality gap approaches 0 as k tends to infinity. It also demonstrates that the optimality gap of the approximation algorithm is far better than the theoretical gap $1/k$. For each value of k, we plot the minimum, average and maximum optimality gap observed over all instances versus the theoretical approximation ratio. These figures depict that when we increase k, both the empirical and theoretical performances of the algorithms get close to each other. Therefore, the rate of overestimation decreases considerably for large k. The results also indicate that even though the decomposition-based heuristic does not prove a theoretical performance bound, the optimality gaps are lower than the theoretical gap provided by the approximation algorithm. However, the maximum optimality gaps of the decomposition-based heuristic are slightly larger than that of the approximation algorithm.

Figure 4.6 compares the performances of `CPLEX` and the approximation algorithm in terms of solution time for different values of k. The approximation algorithm can return a feasible solution with a much less computational effort for many k values compared to `CPLEX`. Recall that k determines the number of levels in each subproblem so it affects the subproblem size. As expected, the empirical performance of the algorithm in terms of solution quality increases with k at the expense of high computation times. Therefore, it is very critical to determine the best value of k. The value of k should be large enough for good approximation but it should be less than a threshold value not to exceed the solution time of `CPLEX`. Figure 4.6 indicates that for small size instances, `CPLEX` outperforms the approximation algorithm when k is larger than 7. Those instances are solved to optimality within the time limit. Therefore, we especially focus on large problems for which `CPLEX` could not find an optimal solution within the time limit. Figure 4.6demonstrates that for k values larger than 20, the solution time for `CPLEX` is less than that of the approximation algorithm. Overall, $k = 20$ seems like a compromise value for this set of instances.

Figure 4.7 compares the performances of the approximation algorithm and the

decomposition-based heuristic in terms of both solution quality and computation time (in seconds). Despite the fact that our heuristic does not guarantee a performance bound, the results demonstrate that the optimality gaps could lower than the theoretical gap. As seen in Figure 4.7a, the solution quality is comparable to that of the approximation algorithm. Since the number of feasible solutions computed by the approximation algorithm increases with k, we need to invest much more computational effort for the approximation algorithm than the decomposition-based heuristic. Therefore, as seen in Figure 4.7b, the approximation algorithm is clearly outperformed by the decomposition-based heuristic in terms of solution time. The decomposition-based heuristic returns a feasible solution whose solution time is $1/k$ that of the approximation algorithm. Therefore, as seen in Figure 4.7b, increasing k also increases the performance gap between the approximation algorithm and the heuristic. The details of the experimental results can be seen in Tables B.8 and B.10 in Appendix B.

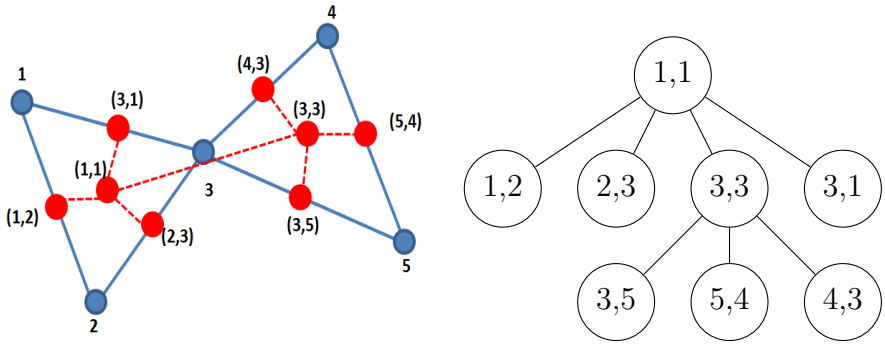
4.1.2 Dynamic Programming Algorithm

In this section, we discuss a DP algorithm to solve both 1-level and more generalized k-level outerplanar graphs, which are obtained by planar graph decomposition technique aforementioned in Section 4.1.1. The algorithm is $O(8^k n)$-time algorithm [7]. If $k = \lceil c \log \log n \rceil$ or $k = \lceil c log n \rceil$, where n is the number of nodes and c is some constant, we get a polynomial time approximation algorithm.

DP Algorithm for Solving 1-Level Outerplanar Graphs We use the tree representation of a planar embedding of a graph and a modified version of the DP algorithm proposed in [7] to solve the MHC problem on outerplanar graphs to optimality. Each outerplanar graph can be represented as a rooted tree $\bar{G}$ such that each leaf node represents an external edge and every other node of the tree represents a face of the outerplanar graph. Throughout the section, we use node for the tree and vertex for the graph to avoid confusion. Here are the steps to obtain $\bar{G}$ from G.

Algorithm 4.2 Algorithm to Convert a Planar Graph into a Rooted Tree

1: Replace each *bridge* (x, y) by two edges between x and y to convert it to a face. A bridge is an edge whose deletion disconnects the graph.
2: Put a node in each interior face and on each external edge of the graph.
3: Draw an edge from each face node that represents an internal face f to either an adjacent face or an exterior edge.
4: Label a leaf node (x, y) as the vertices x and y that are incident to edge (x, y).
5: Label a face node as the first and last nodes in its children's labels. Select one of the face node as a root node and label it as $x = y$.
6: Write the children of a face node in a sequence as a directed walk of exterior edges in a counterclockwise direction from x and y.
7: If the graph includes a *cutpoint*, draw an edge between two face nodes that share the cutpoint. Cutpoint (vertex 3 in Figure 4.8a) is a vertex whose deletion disconnects the graph. In this case, there are some nodes labeled (x, y) other than the root node such that $x = y$. The label means that there are two faces sharing a cutpoint x.



(a) (b)

Figure 4.8: An outerplanar graph and the corresponding tree

Figure 4.8a and Figure 4.8b show an outerplanar graph and its tree representation. The algorithm starts from the leaf nodes and iterates until reaching the root node, (1,1) in Figure 4.8b. At each iteration, the complementary problem MTS is solved for the subgraph rooted at (x, y) due to its simplicity to track the feasibility. DP table keeps a set of bit pairs for each node (x, y) in which represents whether node x and y are included in MTS or not. Also, it keeps the number of nodes in MTS for the subgraph that is rooted at (x, y), which is denoted as $V(x, y)$. For instance, Table 4.1 shows a DP table that represents the tree node $(1, 2)$. First row says that only the vertex 1 is in the solution for the subgraph representing the edge (1,2). The cardinality of the

solution is 1 and the solution is feasible.

DP starts from the leaf nodes and continues upwards by merging the tables. At each iteration, it merges the DP tables rooted at (x, y) and (y, z) and obtain a table rooted at (x, z). Suppose (b_1, b_2) and (b_2, b_3) represent the bit pairs in tables (x, y) and (y, z), respectively. The value of the bit pair (b_1, b_3) in table (x, z) is computed as follows:

$$V(b_1, b_3) = \hat{V}(x, y) + V(b_2, b_3) - b_2$$

where $\hat{V}(x, y)$ is the maximum value of all feasible bit pairs (b_1, b_2) in table (x, y) for b_1, $V(b_2, b_3)$ is the value of bit pair (b_2, b_3) in table (y, z) and b_2 is the bit that maximizes $\hat{V}(x, y)$.

Table 4.2 shows how to solve 1-level outerplanar graph to optimality for the MHC problem. The leftmost, middle and the rightmost blocks represent the DP tables rooted at (x, y), (y, z), and (x, z), respectively. The first entry in the table (3,4) points out the bit pair (0,0) obtained by merging the tables $(3, 5)$ and $(5, 4)$. It can be easily seen that $b_1 = 0$, $\hat{V}(3, 5) = 1$, $b_2 = 1$, and $V(0, 0) = 1$. A procedure named *adjust* identifies the feasibility of each bit pair. A bit pair (b_1, b_3) is tagged as infeasible if both (b_1, b_2) and (b_2, b_3) are $(1, 1)$ and the bits b_1, b_2 and b_3 results in a solution such that all the vertices in a triangle are in MTS. The pairs $(1, 0)$ and $(0, 1)$ are also tagged as infeasible for the DP table rooted at (x, x). If the label of a table is (x, x), adjust procedure decreases the value of the bit pair $(1, 1)$ by one to avoid counting the same bit twice. Infeasible entries in the tables are marked by ✗. The maximum value of the root node $(1, 1)$ is 4 which is the cardinality of optimal MTS. Optimal solution is found by backtracking over the DP tables (see colored rows). For example, the root node says that there are four nodes in MTS and 1 is included. Given vertex 1 is selected, Table $(1, 3)$ says that 3 is not included (maximum of line 3 and line 4 in Table $(1, 3)$). The solution vector after backtracking operations is (1,1,0,1,1); that is, the vertices 1,2,4, and 5 are in MTS. The complement of the solution, (0,0,1,0,0), is the optimal solution of the MHC problem.

Table 4.1: DP table for tree node (1,2)

(1,2)			
1	**2**	**V(1,2)**	**Feasible?**
1	0	1	✓
0	1	1	✓
0	0	0	✓
1	1	2	✓

Table 4.2: DP iterations for MHC problem for the tree in Figure 4.8b

(3,5)				**(5,4)**				**(3,4)**			
0	0	0	✓	0	0	0	✓	0	0	1	✓
1	0	1	✓	1	0	1	✓	1	0	2	✓
0	1	1	✓	0	1	1	✓	0	1	2	✓
1	1	2	✓	1	1	2	✓	1	1	3	✗
(3,4)				**(4,3)**				**(3,3)**			
0	0	1	✓	0	0	0	✓	0	0	2	✓
1	0	2	✓	1	0	2	✓	1	0	3	✗
0	1	2	✓	0	1	1	✓	0	1	1	✗
1	1	3	✗	1	1	1	✓	1	1	2	✓
(1,2)				**(2,3)**				**(1,3)**			
0	0	0	✓	0	0	0	✓	0	0	1	✓
1	0	1	✓	1	0	1	✓	1	0	2	✓
0	1	1	✓	0	1	1	✓	0	1	1	✓
1	1	2	✓	1	1	2	✓	1	1	3	✗
(1,3)				**(3,3)**				**(1,3)**			
0	0	1	✓	0	0	2	✓	0	0	2	✗
1	0	2	✓	1	0	3	✗	1	0	3	✓
0	1	1	✓	0	1	1	✗	0	1	3	✓
1	1	3	✗	1	1	2	✓	1	1	4	✗
(1,3)				**(3,1)**				**(1,1))**			
0	0	2	✗	0	0	0	✓	0	0	3	✓
1	0	3	✓	1	0	2	✗	1	0	4	✗
0	1	3	✓	0	1	1	✓	0	1	3	✗
1	1	4	✗	1	1	1	✓	1	1	4	✓

DP Algorithm for Solving k-Outerplanar Graphs In the previous section, we discussed a DP algorithm to solve the MHC problem on outerplanar graphs to optimality. Here, we generalize the algorithm for k-outerplanar graphs. How to partition the graph affects the size of the DP tables so Baker [7] proposes the decomposition technique that ensures that the size of the table does not exceed 2^{2k}. Therefore, we partition the graph into slices in such a way that each slice has at most $2k$ boundary nodes, one node for each level. Then, we iteratively merge the slices until obtaining the whole graph.

Similar to an outerplanar graph, a k-outerplanar graph can also be represented as a rooted tree. However, how the root and the leftmost child are selected for a tree at level $i > 1$ have to be discussed. Suppose, a level i component C is enclosed by a level $i-1$ face. Then, the root node of level i and the leftmost child are determined based on a triangulation between level i and level $i-1$. For example Figure 4.9a and 4.9b demonstrate a 2-outerplanar graph and triangulations between the first and the second level. Suppose the node at level $i-1$ has a label (x, y), then the root node of the tree at level i would be (z, z) where z is adjacent to x in tree. If $x = y$ then the z can be any vertex adjacent to x, otherwise z is the third vertex of the triangle formed by x and y. For example, suppose C is the level 2 component with the vertices 1-5 which enclosed by the face f, including the vertices A to D in Figure 4.9b. If the label of the root node of the first level is (A, B), then the label of root node of the second level tree would be (3,3). If it is (A, A), then the label of the root at the second level can be any of the pairs (1,1), (2,2) or (3,3). If C includes a single vertex, then the tree includes only the root, otherwise the leftmost child is (z, u) where (z, u) is the first exterior edge of C counterclockwise around z from (z, x) in the tree.

DP algorithm iteratively merges the tables of subgraphs called *slices*. For each node in a tree, there is a level-i slice including the nodes in its label as well as some boundary nodes, one for each level from 1 to i. However, a slice at a level may include higher level boundary vertices when the vertices at that level encloses a component from higher levels. As we mentioned, DP algorithm merges the tables of slices until reaching the root node at the first level where the original graph is obtained. Figure 4.11

illustrates the slices of the nodes labeled as (1,2) and (3,1). Dashed lines represent the connection to the boundary nodes when there does not exist any edge to the boundary nodes. Here, we give the informal definition of the slices. Let v be a tree node labeled (x, y).

(a) If v represents a level i face with no enclosed vertices, $i \geq 1$, its slice is the union of the slices of its children, plus (x, y).

(b) If v represents a level i face enclosing a level $i + 1$ component C, its slice is that of the root of the tree for C plus (x, y).

(c) If v represents a level 1 leaf, its slice is the subgraph consisting of (x, y).

(d) If v represents a level i leaf, its slice includes (x, y), edges from x and y to a level i nodes and the slices computed recursively for appropriate level i trees (appropriate means no edge cross the slice boundaries)

Baker [7] defines a function that determines which boundary nodes will be included for each tree node. A details about the formal definition of the slices in terms of the function value value can be seen in [7]. Slice definition provides that two successive slices that will be merged share a common boundary, i.e., the right boundary of a tree node is equal to the left boundary of the successive tree node, so the original graph is obtained by combining the slices like building a puzzle. Figure 4.1.2 shows the slices of each tree node. The slice of the root node at the first level is the original graph.

Here, we discuss how to solve the MHC problem on a $k-$level outerplanar graph to optimality. DP table of an i-level slice keeps at most 2^{2i} elements, one for each subset of $2i$ boundary nodes. The value of the table is either the number of vertices in MTS including the subgraph represented by that slice or infeasible if the solution violates the feasibility. The table includes one bit for each boundary node and one bit for the value of the table. For example, the first block in Table 4.3, keeps the all possible combinations of the values that the vertex 3, C or 5 can take. The first and next two entries of a block represent the bit pairs that shows whether the nodes in left and the right boundaries (3-C and 5-C) of a slice in MTS or not. The fifth column in

a block shows the cardinality of MTS for the subgraph represented by that slice. For example, fourth entry of the table $(3, C, 5, C)$ says that only the vertex 5 is in MTS. Since only 5 is in the optimal solution, the value of the table is 1. While merging the table, we use the same formula described in the previous section to compute the value of merged table. However, b_1, b_2 and b_3 are the set of boundary nodes rather than individual vertices. Let's calculate the third entry of the slice (3,4) represented by the table $(3, C, 4, D)$. First we search the table $(3, C, 5, C)$ that maximizes the value of that table when $b_1 = (0, 1)$. The maximum value of the table is 2 and b_2 that maximizes the table is $(1, 1)$ (15^{th} entry of table $(3, C, 5, C)$). We merge (0,1,1,1) and (1,1,0,0) and obtain the third entry (0,1,0,0). The value of the entry (0,1,0,0) is equal to the value of the table $(3, C, 5, C)$ for the entry $(0, 1, 1, 1)$ plus the value of the table $(5, C, 4, D)$ for the entry $(1, 1, 0, 0)$ minus 2 (since the number of vertices included in b_2 is 2). The largest entry in table $(1, A, 1, A)$ gives the optimal MTS. The vertices in the optimal MHC can be found as $3, B, C$ by following the underlined entries and taking the complement of MTS.

Algorithm 4.3 summarizes the iterations of the approximation algorithm when the subproblems are solved by the DP algorithm. Notice that Algorithms 4.1 and 4.3 are very similar. The only difference is that the latter implements the DP algorithm rather than solving the IP model (3.1)-(3.3) by `CPLEX` solver.

Table 4.3: DP iterations for merging tables $(3, C, 5, C)$ and $(5, C, 4, D)$ to obtain table $(3, C, 4, D)$

(3,C,5,C)						**(5,C,4,D)**						**(3,C,4,D)**					
0	0	0	0	0	✓	0	0	0	0	0	✓	0	0	0	0	1	✓
1	0	0	0	1	✓	1	0	0	0	1	✓	1	0	0	0	2	✓
0	1	0	0	1	✗	0	1	0	0	1	✓	0	1	0	0	2	✓
0	0	1	0	1	✓	0	0	1	0	1	✓	0	0	1	0	2	✓
0	0	0	1	1	✗	0	0	0	1	1	✓	0	0	0	1	2	✓
1	1	0	0	2	✗	1	1	0	0	2	✓	1	1	0	0	3	✓
1	0	1	0	2	✓	1	0	1	0	2	✓	1	0	1	0	3	✗
1	0	0	1	2	✗	1	0	0	1	2	✓	1	0	0	1	3	✓
0	1	1	0	2	✗	0	1	1	0	2	✓	0	1	1	0	3	✓
0	1	0	1	1	✓	0	1	0	1	2	✓	0	1	0	1	3	✓
0	0	1	1	2	✗	0	0	1	1	2	✓	0	0	1	1	3	✓
1	1	1	0	3	✗	1	1	1	0	3	✓	1	1	1	0	4	✗
1	0	1	1	3	✗	1	0	1	1	3	✓	1	0	1	1	4	✗
1	1	0	1	2	✓	1	1	0	1	3	✓	1	1	0	1	4	✗
0	1	1	1	2	✓	0	1	1	1	3	✓	0	1	1	1	4	✓
1	1	1	1	3	✓	1	1	1	1	4	✓	1	1	1	1	5	✗

Table 4.4: DP iterations for merging tables $(3, C, 4, D)$ and $(4, D, 3, B)$ to obtain table $(3, C, 3, B)$

(3,C,4,D)						**(4,D,3,B)**						**(3,C,3,B)**					
0	0	0	0	1	✓	0	0	0	0	0	✓	0	0	0	0	3	✓
1	0	0	0	2	✓	1	0	0	0	1	✓	1	0	0	0	3	✗
0	1	0	0	2	✓	0	1	0	0	1	✓	0	1	0	0	3	✓
0	0	1	0	2	✓	0	0	1	0	1	✓	0	0	1	0	4	✗
0	0	0	1	2	✓	0	0	0	1	1	✓	0	0	0	1	4	✓
1	1	0	0	3	✓	1	1	0	0	2	✓	1	1	0	0	3	✗
1	0	1	0	3	✗	1	0	1	0	2	✓	1	0	1	0	3	✓
1	0	0	1	3	✓	1	0	0	1	2	✓	1	0	0	1	4	✗
0	1	1	0	3	✓	0	1	1	0	2	✓	0	1	1	0	4	✗
0	1	0	1	3	✓	0	1	0	1	2	✓	0	1	0	1	4	✓
0	0	1	1	3	✓	0	0	1	1	2	✓	0	0	1	1	5	✗
1	1	1	0	4	✗	1	1	1	0	3	✓	1	1	1	0	4	✓
1	0	1	1	4	✗	1	0	1	1	3	✓	1	0	1	1	5	✓
1	1	0	1	4	✗	1	1	0	1	3	✓	1	1	0	1	4	✗
0	1	1	1	4	✓	0	1	1	1	3	✓	0	1	1	1	5	✗
1	1	1	1	5	✗	1	1	1	1	4	✓	1	1	1	1	5	✓

Table 4.5: DP iterations for merging tables $(1, A, 2, A)$ and $(2, A, 3, C)$ to obtain table $(1, A, 3, C)$

(1,A,2,A)						**(2,A,3,C)**						**(1,A,3,C)**					
0	0	0	0	0	✓	0	0	0	0	0	✓	0	0	0	0	1	✓
1	0	0	0	1	✓	1	0	0	0	1	✓	1	0	0	0	2	✓
0	1	0	0	1	✗	0	1	0	0	1	✓	0	1	0	0	2	✓
0	0	1	0	1	✓	0	0	1	0	1	✓	0	0	1	0	2	✓
0	0	0	1	1	✗	0	0	0	1	1	✓	0	0	0	1	2	✓
1	1	0	0	2	✗	1	1	0	0	2	✓	1	1	0	0	3	✓
1	0	1	0	2	✓	1	0	1	0	2	✓	1	0	1	0	3	✗
1	0	0	1	2	✗	1	0	0	1	2	✓	1	0	0	1	3	✓
0	1	1	0	2	✗	0	1	1	0	2	✓	0	1	1	0	3	✓
0	1	0	1	1	✓	0	1	0	1	2	✓	0	1	0	1	3	✓
0	0	1	1	2	✗	0	0	1	1	2	✓	0	0	1	1	3	✓
1	1	1	0	3	✗	1	1	1	0	3	✓	1	1	1	0	4	✗
1	0	1	1	3	✗	1	0	1	1	3	✓	1	0	1	1	4	✗
1	1	0	1	2	✓	1	1	0	1	3	✓	1	1	0	1	4	✗
0	1	1	1	2	✓	0	1	1	1	3	✓	0	1	1	1	4	✓
1	1	1	1	3	✓	1	1	1	1	4	✓	1	1	1	1	5	✗

Table 4.6: DP iterations for merging tables $(1, A, 3, C)$ and $(3, C, 3, B)$ to obtain table $(1, A, 3, B)$

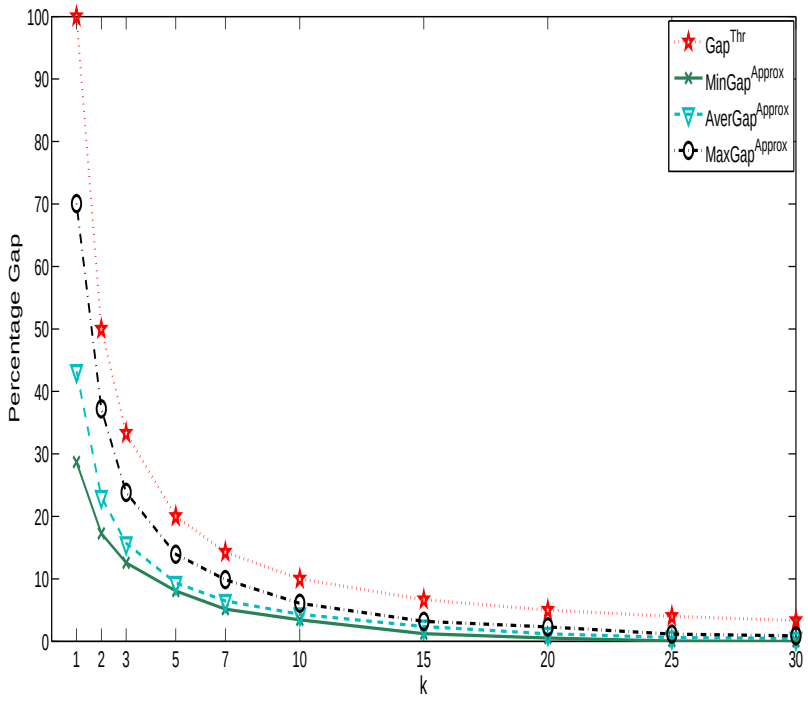
(1,A,3,C)						**(3,C,3,B)**						**(1,A,3,B)**					
0	0	0	0	1	✓	0	0	0	0	3	✓	0	0	0	0	4	✗
1	0	0	0	2	✓	1	0	0	0	3	✗	1	0	0	0	5	✓
0	1	0	0	2	✓	0	1	0	0	3	✓	0	1	0	0	5	✗
0	0	1	0	2	✓	0	0	1	0	4	✗	0	0	1	0	4	✓
0	0	0	1	2	✓	0	0	0	1	4	✓	0	0	0	1	5	✗
1	1	0	0	3	✓	1	1	0	0	3	✗	1	1	0	0	6	✓
1	0	1	0	3	✗	1	0	1	0	3	✓	1	0	1	0	6	✗
1	0	0	1	3	✓	1	0	0	1	4	✗	1	0	0	1	6	✗
0	1	1	0	3	✓	0	1	1	0	4	✗	0	1	1	0	5	✓
0	1	0	1	3	✓	0	1	0	1	4	✓	0	1	0	1	6	✗
0	0	1	1	3	✓	0	0	1	1	5	✗	0	0	1	1	5	✓
1	1	1	0	4	✗	1	1	1	0	4	✓	1	1	1	0	7	✗
1	0	1	1	4	✗	1	0	1	1	5	✓	1	0	1	1	7	✗
1	1	0	1	4	✗	1	1	0	1	4	✗	1	1	0	1	7	✗
0	1	1	1	4	✓	0	1	1	1	5	✗	0	1	1	1	7	✗
1	1	1	1	5	✗	1	1	1	1	5	✓	1	1	1	1	8	✗

Table 4.7: DP iterations for merging tables $(1, A, 3, B)$ and $(3, B, 1, A)$ to obtain table $(1, A, 1, A)$

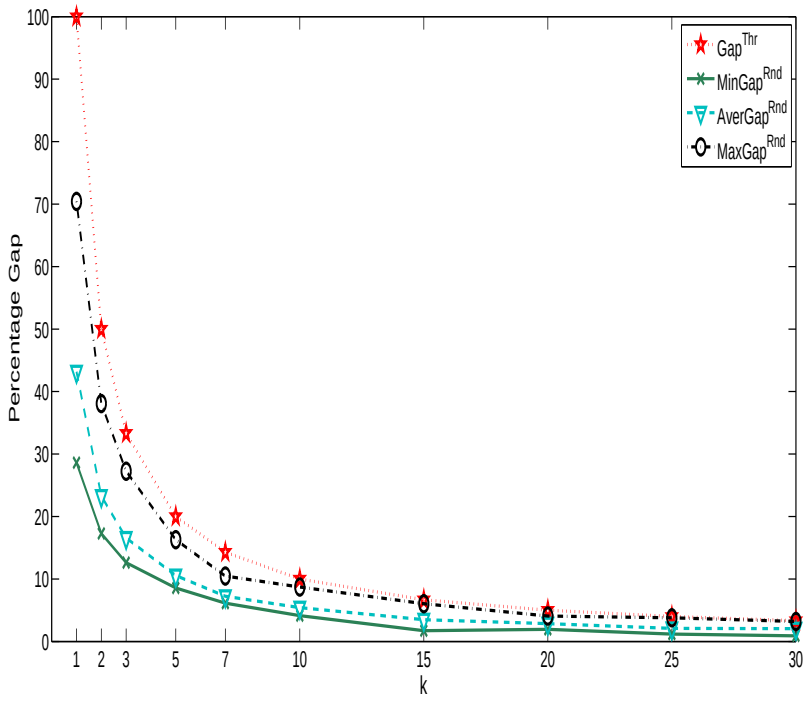
(1,A,3,B)						**(3,B,1,A)**						**(1,A,1,A)**					
0	0	0	0	4	✗	0	0	0	0	0	✓	0	0	0	0	4	✓
1	0	0	0	5	✓	1	0	0	0	1	✓	1	0	0	0	5	✗
0	1	0	0	5	✗	0	1	0	0	1	✓	0	1	0	0	5	✓
0	0	1	0	4	✓	0	0	1	0	1	✓	0	0	1	0	5	✗
0	0	0	1	5	✗	0	0	0	1	1	✓	0	0	0	1	5	✗
1	1	0	0	6	✓	1	1	0	0	2	✓	1	1	0	0	6	✗
1	0	1	0	6	✗	1	0	1	0	2	✓	1	0	1	0	5	✓
1	0	0	1	6	✗	1	0	0	1	2	✓	1	0	0	1	6	✗
0	1	1	0	5	✓	0	1	1	0	2	✓	0	1	1	0	6	✗
0	1	0	1	6	✗	0	1	0	1	2	✓	0	1	0	1	6	✓
0	0	1	1	5	✓	0	0	1	1	2	✓	0	0	1	1	6	✗
1	1	1	0	7	✗	1	1	1	0	3	✓	1	1	1	0	7	✗
1	0	1	1	7	✗	1	0	1	1	3	✓	1	0	1	1	7	✗
1	1	0	1	7	✗	1	1	0	1	3	✓	1	1	0	1	7	✗
0	1	1	1	7	✗	0	1	1	1	3	✓	0	1	1	1	7	✗
1	1	1	1	8	✗	1	1	1	1	4	✓	1	1	1	1	6	✓

Algorithm 4.3 DP Algorithm for Planar Graphs

Input: A planar embedding of G, the vertices lying in each level, and k
Output: HC^{approx}
$HC^i \leftarrow \emptyset$ for each $i \in \{1, 2, \cdots, k\}$
$HC^{approx} \leftarrow \emptyset$
for $i := 1$ **to** k **do**
    $j \leftarrow 0$
    while $(j+1)k + i$th level of G is available **do**
        Obtain the subgraph, S_j^i, induced by levels $jk + i$ to $(j+1)k + i$
        Find the tree representation of S_j^i
        Obtain S_j^{*i} by implementing DP algorithm to S_j^i
        $HC^i = HC^i \cup S_j^{*i}$
        $j \leftarrow j + 1$
    end while
end for
Return $HC^{approx} \leftarrow HC^p$, where $HC^p = \arg\min\{|HC^i| \ |1 \leq i \leq k\}$



(a) Approximation Algorithm



(b) Decomposition-based Heuristic

Figure 4.5: Observed vs. theoretical approximation gaps obtained by both the approximation algorithm and the decomposition-based heuristic

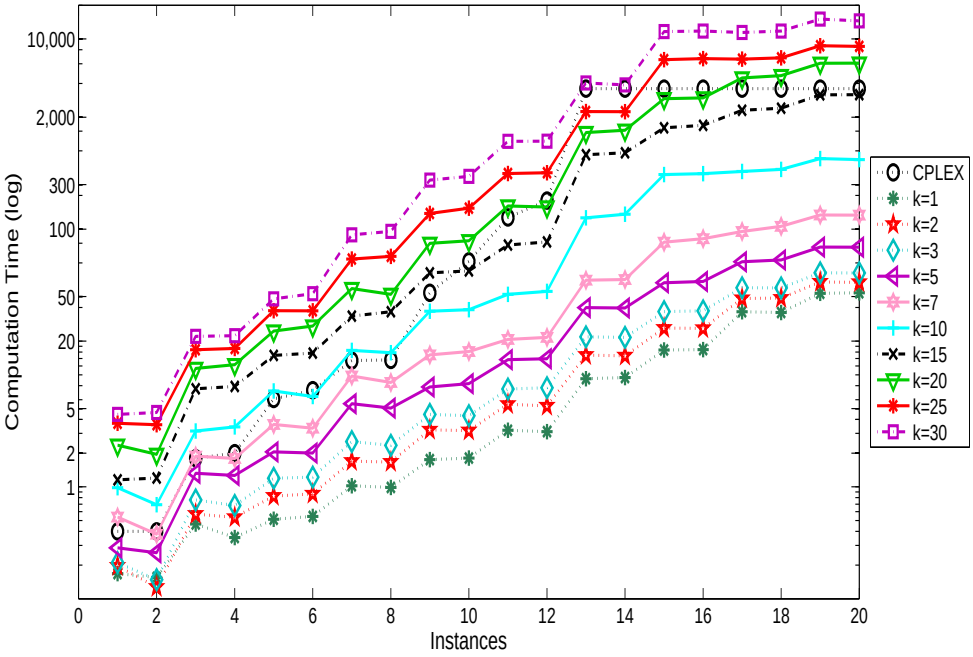
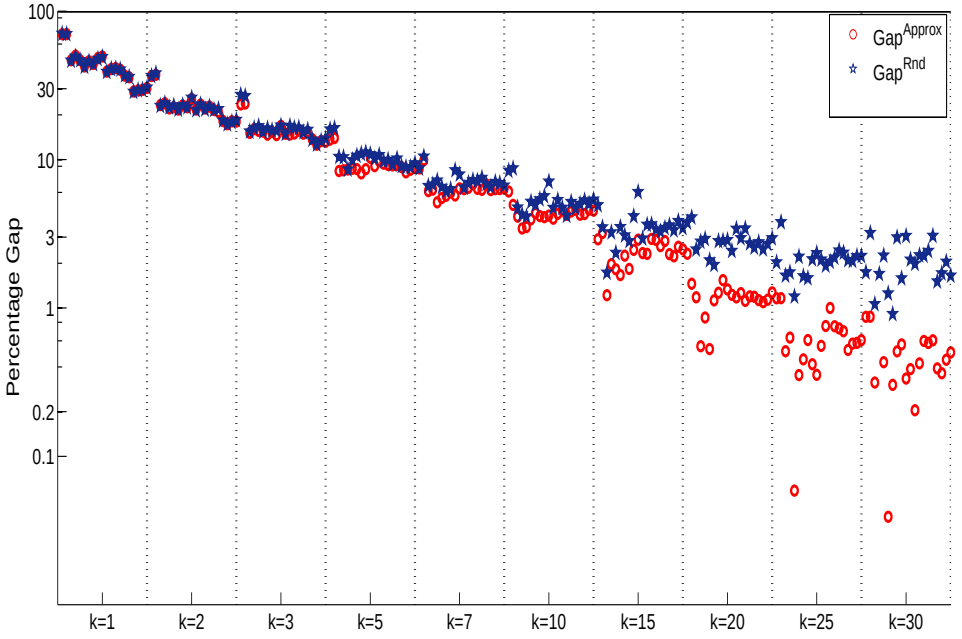
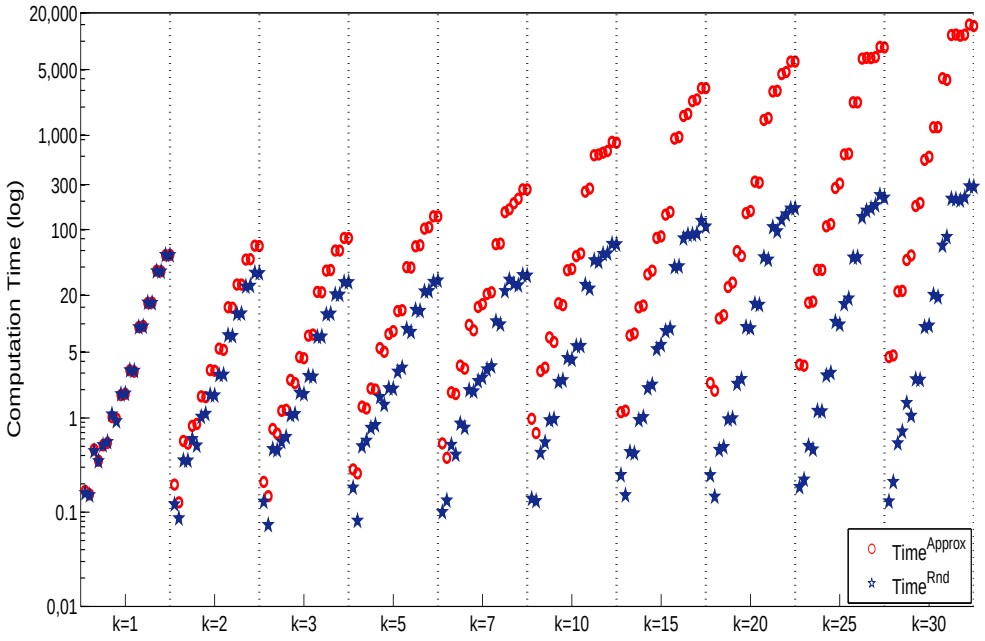


Figure 4.6: Computation times of the approximation algorithm and `CPLEX`



(a) Optimality gaps.



(b) Computation time.

Figure 4.7: Percentage gaps and computation times of the approximation algorithm and the decomposition-based heuristic

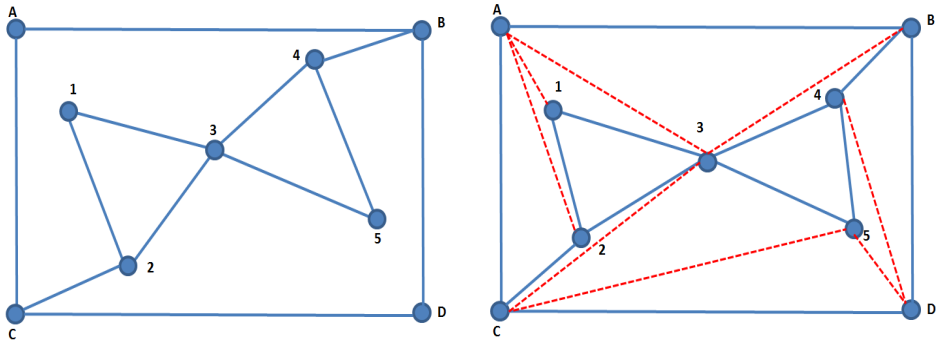


Figure 4.9: A 2-level outerplanar graph and possible triangulations shown as dashed lines

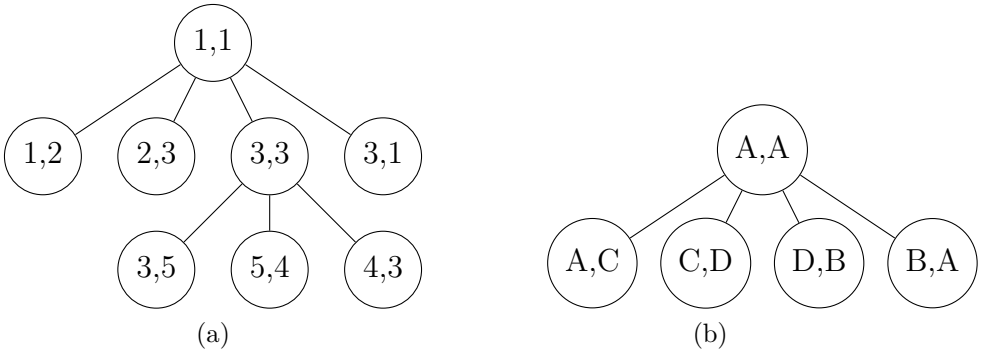


Figure 4.10: Trees for the graph in Figure 4.9a

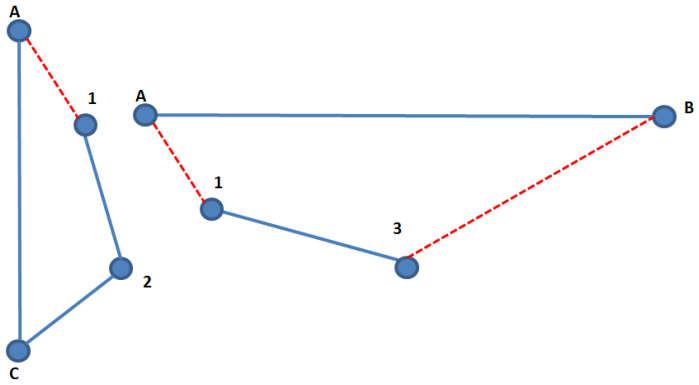
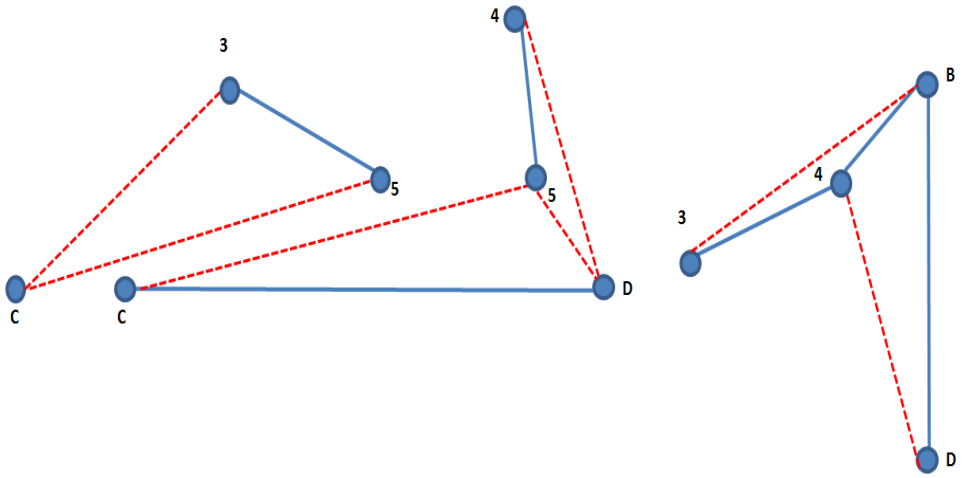
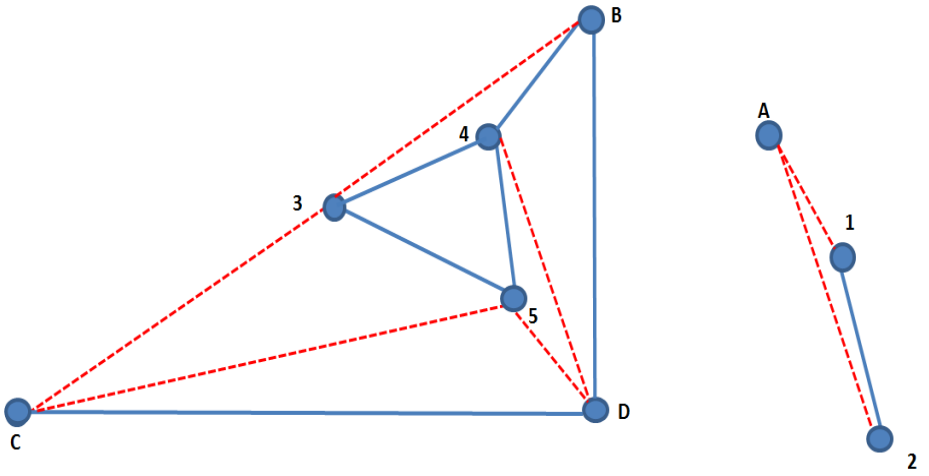


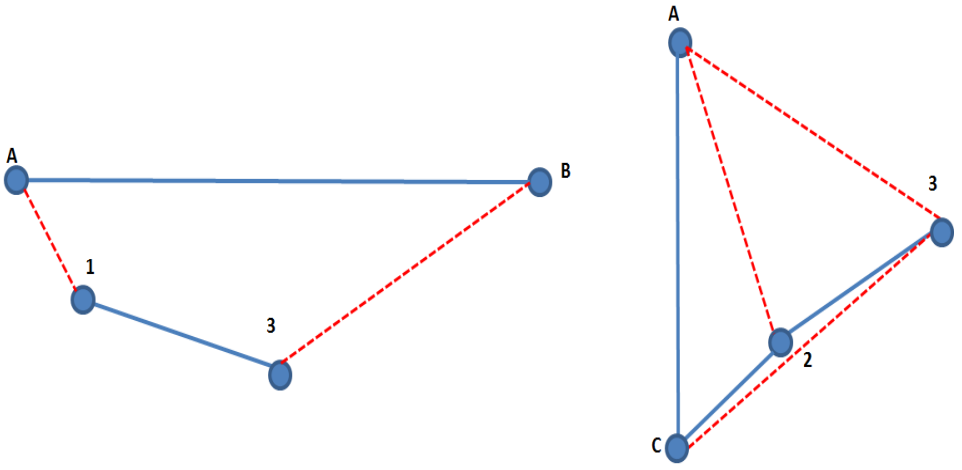
Figure 4.11: The slices for the second level tree nodes (1,2) and (3,1) for the graph in Figure 4.9a



(a) Slices $(3, C, 5, C)$, $(5, C, 4, D)$, and $(4, D, 3, B)$



(b) Slices $(3, C, 3, B)$ and $(1, A, 2, A)$



(c) Slices $(1, A, 3, B)$ and $(2, A, 3, C)$

Figure 4.12: The slices at all level and for each tree node

4.2 Greedy Algorithms

In this section, we discuss two greedy algorithms, which are adapted from the vertex cover literature. These algorithms return a feasible solution very quickly. We compare the solutions of the greedy algorithms against the solutions obtained by a mathematical programming-based heuristic originally proposed for the set covering problem.

Exact algorithm: The IP formulation (3.1)-(3.3) is solved by an off-the-shelf solver to optimality. Since the MHC problem is shown to be $\mathcal{NP}$-Hard, this approach may have practical value only for small-to-medium-scale graphs. However, it sets a definitive benchmark for comparing the performances of various heuristics.

Approximation and greedy algorithms: We implemented two different approximation algorithms. First algorithm selects the vertex with the highest degree at each iteration. The degree of a vertex is defined as the number of adjacent neighbours. The aim is to cover as many edges as possible. Next, all covered edges as well as the vertices in the cover are removed from the graph. The algorithm ends when there is no uncovered edge in the graph. The algorithm is called the $H(\Delta)$-*approximation algorithm* (GR1) for the MVC problem. Here, Δ is the maximum degree in the graph, and $H(\Delta)$ is evaluated by

$$H(\Delta) = 1 + 1/2 + \ldots + 1/\Delta.$$

The second algorithm (GR2), the *2-approximation algorithm*, is an adaptation of [11] originally proposed for computing a near-optimal solution for the MVC problem. Unlike the previous algorithm, it selects an edge arbitrarily, then both vertices incident to that edge are added to the cover.

Mathematical programming-based heuristics: Yelbay et al. [109] propose a heuristic (MBH) that uses the dual information obtained from the LP relaxation of the IP model of SCP. They show the efficacy of the heuristic on a large set of SCP instances. In their work, the dual information is used to identify the most promising columns and then form a restricted problem with those columns. Then, an integer

feasible solution is found by one of the two approaches. In the first approach (MBH), the exact IP optimal solution is obtained by solving *the restricted problem*. In the second approach, a METARAPS [71] local search heuristic (LSLP) is applied over those promising columns. We use both of these approaches.

Experimental Setup. The NP-completeness of MHC speaks to the hardness of the solvability in general. In this section, our goal is to design a set of experiments using different graph types, size and graph density parameters to study how the solvability and the quality of MHC solutions depend on these parameters. The goal is to experimentally identify problem classes for which available solutions are practical and acceptable, and the classes for which new heuristic solutions are warranted. We therefore employ different algorithms to show the trade-offs between optimality and computation time (in seconds) over different graph types.

The optimal LP and IP solutions are obtained by `ILOG IBM CPLEX 12.4` on a personal computer with an Intel Core 2 Dual processor and 3.25 GB of RAM. In all problem instances, the upper limit on the computation is set at 3,600 seconds. The batch processing of the instances is carried out through simple C++ scripts. Our data set includes a total of 830 instances. We have 5 different instances for each combination of a graph type, size, and density parameter to be able to draw conclusions.

We have chosen to use the benchmark database graph instances in [93] and our own synthetically generated data set for our numerical study. This is a very large database of different graph types and sizes designed specifically to test the sophistication of (sub)graph isomorphism algorithms. Since we are using subgraph isomorphism as a basic vehicle for graph matching, the instances selected are thus representative of the class of queries we are likely to handle when we solve the MHC problem. The descriptions of the graph instances we have chosen from this collection are listed below.

a) **Randomly connected graphs:** A *random graph* is a graph that is generated by a model that produces a probability distribution on the graph. In the literature, there are several random graph models that produce different probability distributions. Among these, we have two closely related Erdos Renyi random graph

models [8, 15]. In these models, the probability of having an edge between two vertices is the same for all edges. Therefore, the generated graphs have no special structure. In this class, the number of vertices range from 20 to 1000 ($|V|$=20, 60, 100, 200, 600, 1000). The parameter η denotes the probability of having an edge between any pair of vertices. Thus, this parameter, in a sense, specifies the sparsity of a graph. In the database, three different values of η (0.01, 0.05, and 0.10) are considered. Our data set includes a set of graphs of different sizes for each value of η.

b) **Bounded valence graphs:** A *bounded graph* or a *regular graph* is a graph such that all the vertices have the same degree (*fixed valence*). The sizes of the instances are similar to those of the problem class (a). We use three different values of valence – 3, 6 and 9 to obtain graphs of different size and valence. Bounded valence graphs are generally employed in the modeling of molecular structures.

c) **Irregular bounded valence graphs:** Some irregularities are added to bounded valence graphs to obtain irregular bounded valence graphs. With this modification, the average number of degree is again bounded but some of the vertices may have higher degrees. The sizes of the instances are similar to those of problem classes (a) and (b).

d) **Regular meshes with 2D, 3D, and 4D:** A *mesh, lattice, or grid graph* is a graph whose drawing is embedded in some Euclidean space $\mathcal{R}^n$. In this drawing, incident vertices of each vertex have the same symmetrical tiling and the number of incident vertices of each vertex is the same. In this class, the numbers of vertices range from 16 to 1024, 27 to 1000, and 16 to 1296, respectively. Similar to the problem classes (a), (b), and (c), we have a set of graphs for each combination of size and dimension. 3D objects can be represented as 3D mesh graphs in object recognition.

e) **Irregular meshes:** Irregularity comes from the addition of a certain number of

edges to the graph. The number of edges added to the graph is $\rho \times |V|$, where $\rho \in \{0.2, 0.4, 0.6\}$. The number of vertices is exactly the same as in problem class (d).

f) **Scale-free graphs:** Many real networks are called as *scale-free networks* such as social, biological, information and World Wide Web links. They follow a power-law distribution of the form

$$P(k) \sim \beta k^{-\alpha},$$

where $P(k)$ is the probability that a randomly selected vertex has exactly k edges, β is the normalization constant, and $2 \leq \alpha \leq 3$ is a fixed parameter. The value of the parameter α is approximately 2.72 for the World-Wide-Web. The network generation process of scale-free networks is fundamentally different from that of the random networks [14]. Batagelj [15] proposes a model based on so called *Preferential Attachment Process*. In this model, new vertices added to the graph are connected preferentially to high degree vertices. The probability of being connected to a given vertex is proportional to its current degree. The model requires to know the degree of each vertex in the graph to calculate the probability of being linked. As an alternate model, Herrera and Saramaki [57, 94] propose models that do not require such a global information. The models use the principle of *random walk*. The vertices that are connected to a new added vertex is determined by a random walk in the graph. The probability that a random walk visits a high degree vertex is higher than that a low degree vertex so the models maintain the power law degree distribution. We employed the scale-free graph generator of C++ Boost Graph Library. The generator (Power Law Out Degree algorithm) takes three inputs. These are the number or vertices, α and β. Increasing the value of β increases the average degree of vertices. On the other hand, increasing the value of α decreases the probability of observing vertices with high degrees. The sizes of the instances range from 20 to 1000; $|V| \in \{20, 60, 100, 200, 600, 1000\}$ to be precise. We considered two values for $\alpha \in \{1.5, 2.5\}$ and three values of $\beta \in \{100 \times |V|, 200 \times |V|, 500 \times |V|\}$. Graphs in social

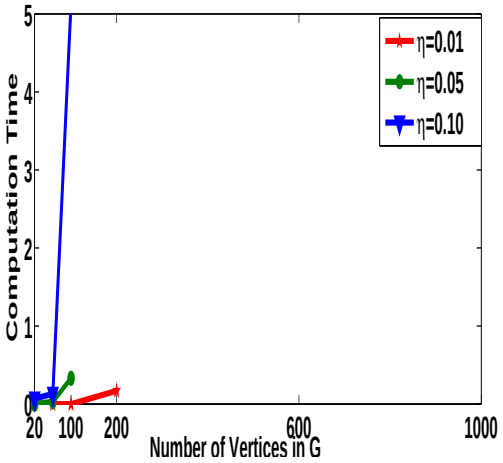
networks, protein-protein interaction networks, flight networks, and computer networks are examples of this class.

We focus on analyzing and understanding the MHC solution methods on the instances above in three different axes: (i) optimal solvability of MHC, (ii) quality of the solutions, and (iii) computational cost of optimal solution. These analyses are aimed at understanding which problem classes are inherently more difficult relative to others so that depending on the application and query, a suitable algorithm can be selected to compute MHC. We also discuss the factors that increase the complexity of the problems. The details of the computational results can be seen in Appendix B, Tables B.1-B.6.

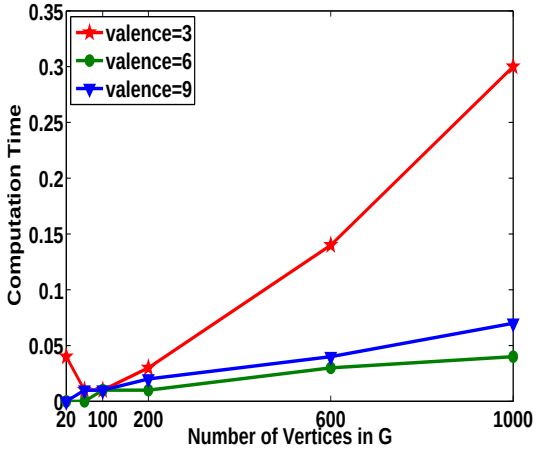
Optimal Solvability of Minimum Hub Covers. Figure 4.12 shows how the optimal solution time of `CPLEX`, an exact method, varies depending on the problem size, class, and structure. The x-axis and the y-axis represent the number of vertices and the average computation time (in seconds), respectively. The right-most data point on a line shows the size of the largest instance that can be solved to optimality in a group. In general, its performance is good for small to medium scale graphs. However, in our study, 39 out of 90, 73 out of 285 and 39 out of 180 instances in problem classes (a), (e) and (f), respectively, could not be solved optimally using `CPLEX` within the time limit. This observation opens the door for heuristics to find acceptable but possibly suboptimal solutions.

From Figure 4.13a we conclude that for randomly connected graphs with more than 200 nodes, optimal solution is not achievable within the bounded time. It also suggests that the density of graphs is a factor that affects the solvability. The solver does increasingly better as the density η goes down (up to 0.01) for the same number of vertices. Its sensitivity with respect to the size and density is apparent in the plots for η equal to 0.05 and 0.10, i.e., a 16 fold increase in solution time.

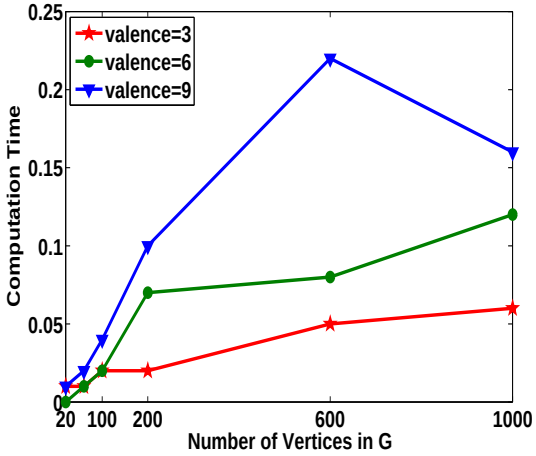
Compared to random graphs, Figure 4.13b shows an improved performance on bounded valence graphs solving all instances under 0.3 seconds. The reason for the performance difference may be due to the considerably higher number of edges in a



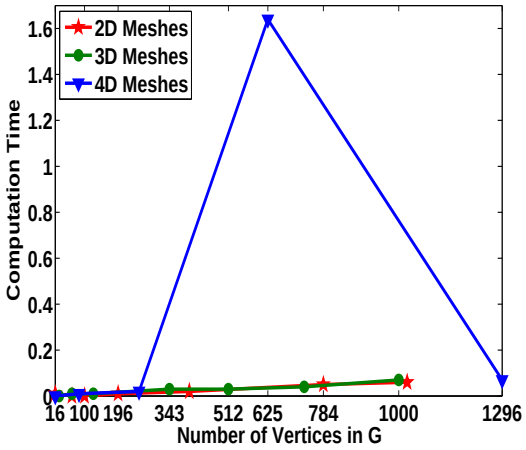
(a) Randomly connected graphs



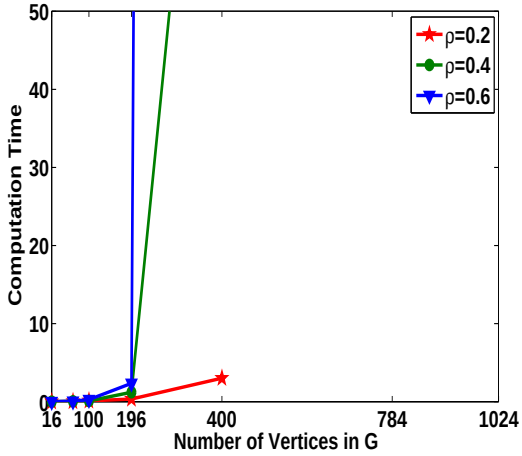
(b) Bounded valence graphs



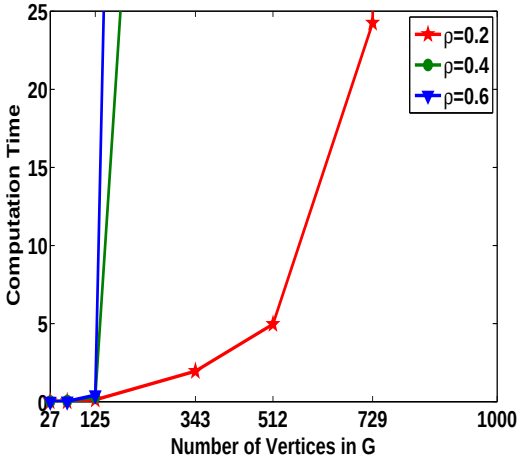
(c) Irregular bounded valence graphs



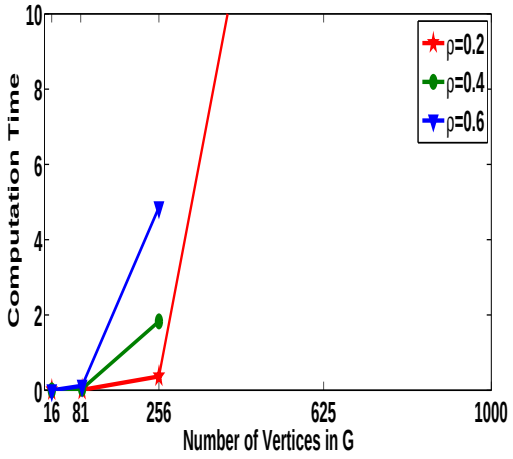
(d) Regular meshes with 2D, 3D and 4D



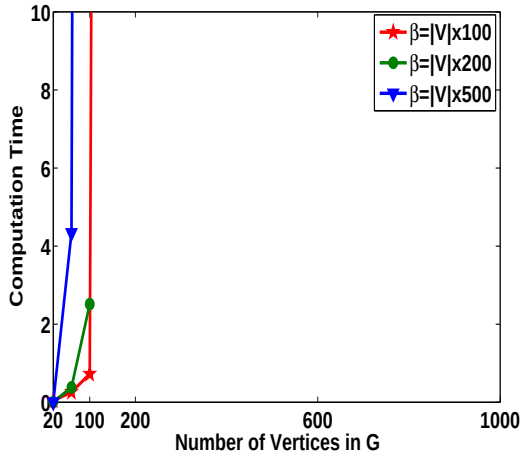
(e) Irregular meshes with 2D



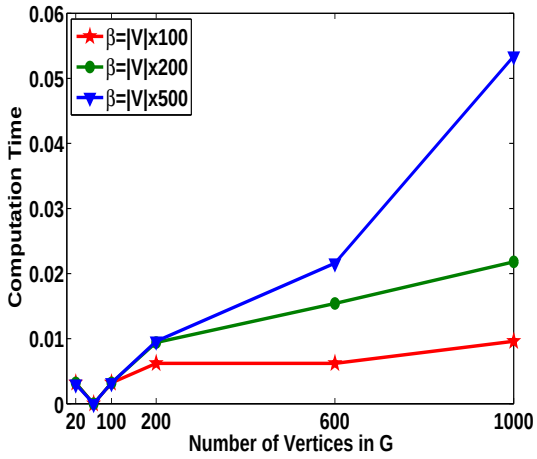
(f) Irregular meshes with 3D



(g) Irregular meshes with 4D



(h) Scale-free graphs with $\alpha = 1.5$



(i) Scale-free graphs with $\alpha = 2.5$

Figure 4.12: Average computation time of CPLEX on problem classes as a function of the number of vertices in G and the parameters of the problem classes

randomly connected graph (which forces the number constraints in the IP model to go higher) than that of a bounded valence graph. However, although we expect higher solution time for graphs with larger valence, Figure 4.13b shows substantially higher time for valence 3 than valences 6 and 9 suggesting other factors may also be playing a role.

Although the degree distribution is neither constant nor fully randomly distributed, for irregular bounded graphs, CPLEX performs similarly to bounded valence graphs. As figure 4.13c shows, all solutions are computed in less than 0.25 seconds, and that the computation time increases with the increase in valence.

Figure 4.13d shows that the size of meshes (2D, 3D or 4D) usually does not have any influence on the performance barring the abrupt behavior of the 4D mesh graph. In general, the solution time appears to linearly increase with the increase in graph size, though the increase in time is extremely small.

Unlike the irregular bounded valence graphs, mesh graphs are more susceptible to irregularity and the computation time substantially increases with the degree of irregularity. Figures 4.13e through 4.12g show that the sizes of the problems that can be solved to optimality decrease and the computation times increase with increasing degree of irregularity. This result is quite reasonable and expected because increasing irregularity increases the number of edges, and thus the computation time as well. This is also because randomly adding edges to a mesh graph makes it structurally more similar to random graphs, which, as discussed earlier, is inherently hard to solve.

We consider the effect of the two parameters α and β on the solvability of the scale-free problems. On one hand, increasing α makes the degree distribution sharper, i.e, we observe smaller number of vertices with high degrees. On the other hand, increasing the value of β increases the degrees of non-hub nodes. Figure 4.12h and Figure 4.12i represent the optimal solution times of scale-free instances. It is clear that the difficulty of the problem is closely related to parameters α and β. The figures show that computation times decrease significantly with increasing values of α. When $\alpha = 1.5$, the instances with more than 100 vertices cannot be solved to optimality within the time limit. When $\alpha = 2.5$, however, all of the instances can be solved optimally

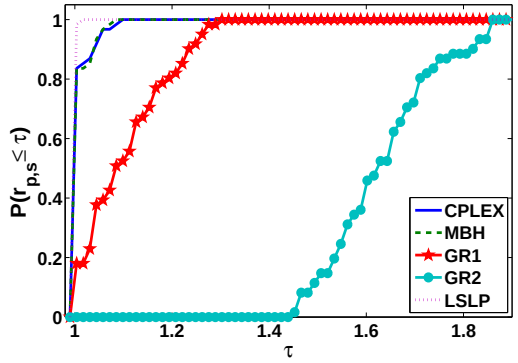
in less than 0.06 seconds. These figures also show that the computation time increases with increasing values of β. This means that increasing degrees of non-hub nodes makes the problem more difficult.

Performance Profile of Solution Methods. To study the quality of solutions generated by other solution methods with respect to the optimal solutions computed using `CPLEX`, we refer to Figures 4.13a through 4.13e. These plots are called performance profiles of algorithms that depict the fraction of problems for which the algorithm is within a factor of the best solution [35]. Thus, they compare the performance of an algorithm s on an instance p with the best performance observed by any other algorithm on the same instance. The x-axis represents the *performance ratio* given by

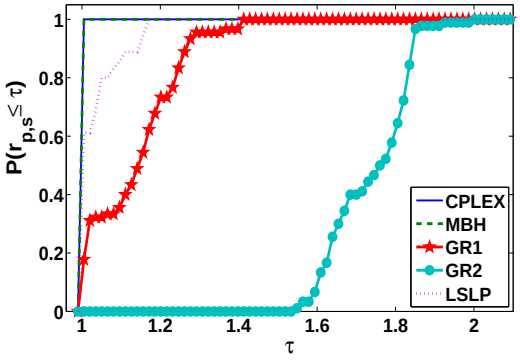
$$r_{p,s} = \frac{\alpha_{p,s}}{\min\{\alpha_{p,s} : s \in S\}},$$

where $\alpha_{p,s}$ is the number of hub nodes in the hub cover when the instance p is solved by algorithm s and S is the set of all benchmark algorithms. The y-axis shows the percentage of the instances that gives a solution that is less than or equal to τ times the best solution. Recall that `CPLEX` cannot solve all of the instances in problem classes (a), (e) and (f) to optimality. However, the solver is able to find feasible solutions for some of those unsolved instances (11 out of 39, 44 out of 73, and 24 out of 39 in (a), (e), and (f) respectively). Figure 4.13 includes all instances except those for which `CPLEX` cannot find either feasible or optimal solutions within the time limit.

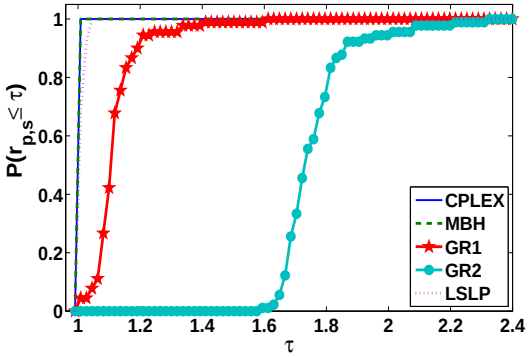
We first analyze how much we sacrifice from the optimality by employing mathematical programming-based heuristics MBH and LSLP. Recall that MBH and LSLP solve the same restricted problem. While MBH tries to solve the problem to optimality, the latter visits alternate solutions in the feasible region. Figure 4.13a shows that 12% of the instances in class (a) where both MBH and LSLP find better feasible solutions than that of `CPLEX`. Note that, this can happen if and only if `CPLEX` returns a feasible solution rather than an optimal solution within the time limit. For other problem classes, the performances of the `CPLEX` and MBH are quite similar. Moreover, these figures show that LSLP is outperformed by MBH and `CPLEX` on almost 40% and 30%



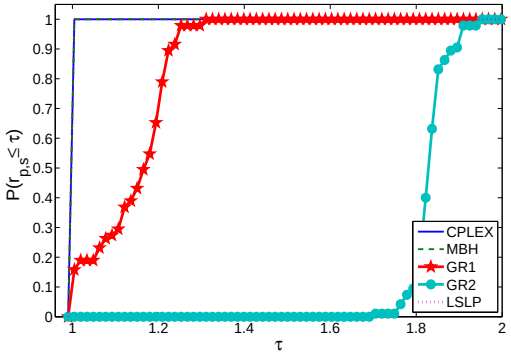
(a) Randomly connected graphs



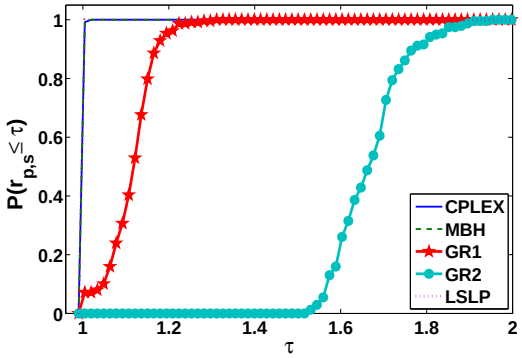
(b) Bounded valence graphs



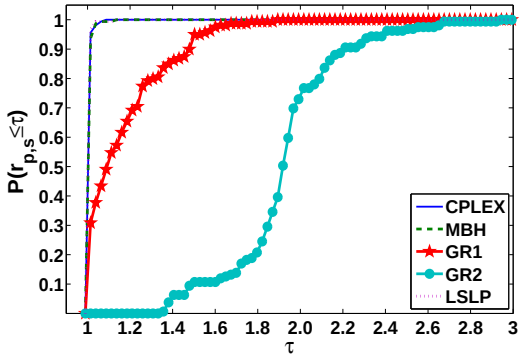
(c) Irregular bounded valence graphs



(d) Regular meshes with 2D, 3D and 4D



(e) Irregular meshes



(f) Scale-free graphs

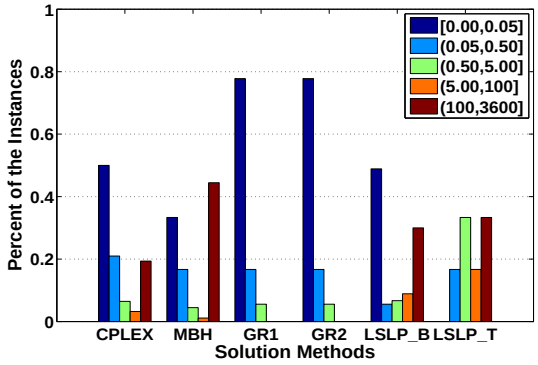
Figure 4.13: Performance profiles for the algorithms on the problem classes in terms of solution quality

of instances in problem classes (b) and (c) respectively. For the remaining problem classes, the performance of LSLP is also comparable to MBH and `CPLEX`.

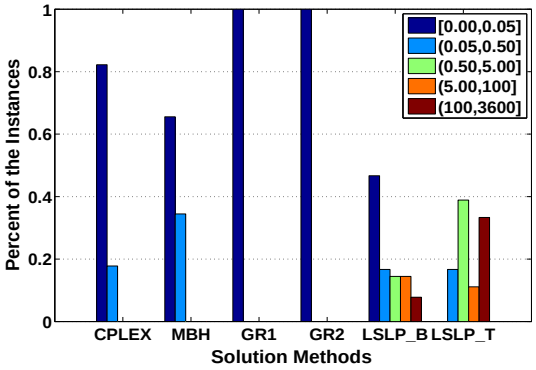
The greedy algorithms return feasible but sub-optimal solutions quickly. Except the scale-free networks, the performances of the greedy algorithms do not change with respect to problem classes. GR2 is known as 2-approximation algorithm for MVC. Figure 4.13c and Figure 4.13f show that there are some instances for which performance ratios of GR2 are higher than 2. Obviously, the approximation ratio of GR1 for MHC problem is higher than 2. Intuitively, the performance of GR1 is supposed to be better when the degree distribution of the vertices is not uniform. Since the average degrees of the vertices are identical or are quite similar for the instances in problem classes (a)-(e), the performance of GR1 does not vary for these problem classes. However, Figure 4.13f shows that GR1 finds the optimal or the best solution in 30% of the scale-free instances. This means that the performance of the GR1 is better for the graphs that follow the power-law distribution.

Cost Profile of Solution Methods. The previous two analyses focused on the optimal solvability and the quality of the solutions. Here we turn our attention to the cost of computing a feasible or optimal MHC solutions in terms of time. Figures 4.14a through 4.14f summarize the distribution of the average computation times (in seconds) of the algorithms over the problem classes. In these plots, the instances for which feasible solutions were not found by any of the algorithm within a time limit are excluded. Each bar in the figure represents the percentage of the instances that are solved within the time interval stated in the legend, e.g., the blue bar for 0.0 to 0.05 seconds. Since LSLP is a local search algorithm, we show both the total computation time and the first time when the best solution is found.

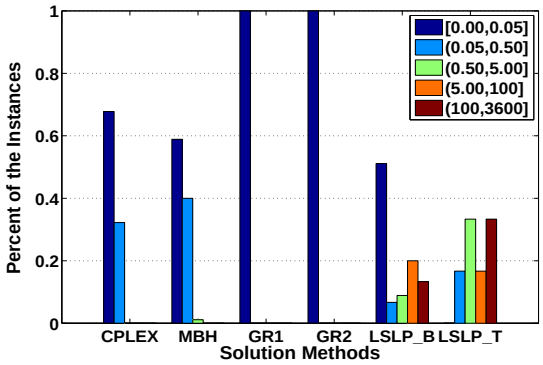
The results clearly show that the solution times of greedy algorithms (GR1, GR2) are much shorter than that of the other algorithms. MBH can solve the restricted problem to optimality in a reasonable amount of time for a great majority of the instances. We have already discussed earlier that the performance of the MBH is good in terms of its solution quality. However, the main drawback for MBH is its inability



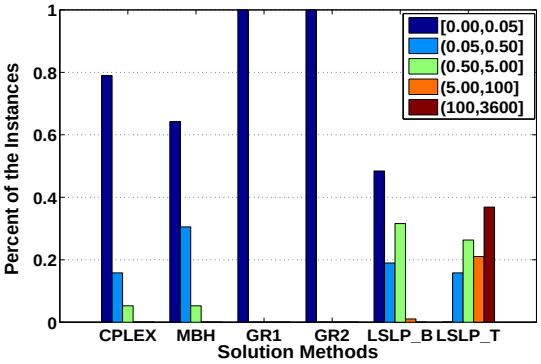
(a) Randomly connected graphs



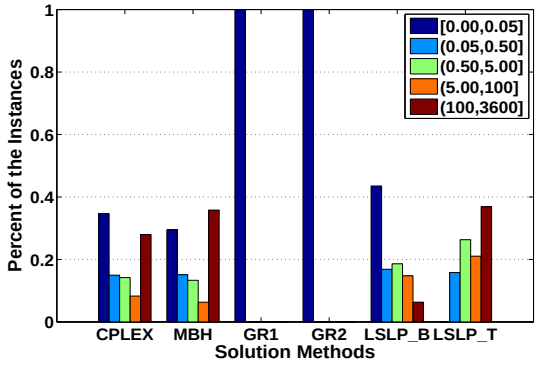
(b) Bounded valence graphs



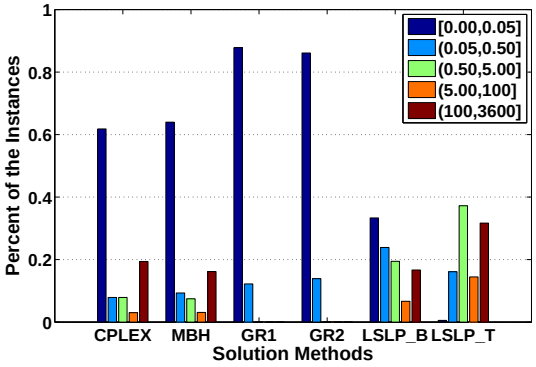
(c) Irregular bounded valence graphs



(d) Regular meshes with 2D, 3D and 4D



(e) Irregular meshes



(f) Scale-free graphs

Figure 4.14: Computation time distributions of the solution methods on the problem classes

to solve the restricted problem to optimality. In such cases, LSLP may serve as an alternative to MBH as it is comparable to MBH in terms of both solution quality and time, and because LSLP is a local search algorithm, it is also guaranteed to produce a feasible solution. However, the performance of LSLP is dependent upon prudent selection of algorithmic parameters, e.g., the total number of iterations, the number of improvement iterations (see [109] for details). There is a trade-off between solution time and the solution quality. Decreasing the total number of iterations may result in a decrease in the total solution time. However, it may increase the optimality gap.

4.3 Relaxation Heuristics

In this section, we first introduce a new rounding algorithm for the MTS problem. As previously mentioned, our first model (3.1)-(3.3) is a special case of the set covering problem. Therefore, we also customize two other rounding algorithms that were originally proposed to solve the set covering problem.

Primal Rounding Algorithm for the MTS Problem ($PRMTS$): The algorithm uses the optimal solution of LP2. The pseudo-code of the algorithm is given in Algorithm 4.4. In line 3, we solve LP2 and obtain the optimal solution, x^*. We select the kth variable, which is the largest component in x^* in line 5. Then, kth vertex is selected and the right hand side of the constraints including that vertex is decreased by 1. The algorithm continues to select the next vertex with largest value as long as none of the constraints is violated.

Primal Rounding Algorithm for the MHC Problem ($PRMHC$): Algorithm 4.5 is adapted from a set covering algorithm proposed by Hochbaum [59]. The algorithm uses the optimal solution of LP1 denoted by x^*. Any component of x^* with value greater than or equal to $1/f$ is set to 1. In the hub cover formulation, f is the maximum number of vertices that can cover an edge. This approach is guaranteed to yield a feasible solution for MHC. Suppose $PRMHC$ does not yield a feasible solution, then there exists at least one constraint for edge (i, j) such that $x_j^* < 1/f$ for all $j \in \bar{\mathcal{K}}^{(i,j)}$.

Algorithm 4.4 Primal Rounding Algorithm for the MTS Problem

$x_j = 0, \quad \forall j \in V$
$y_{ij} \leftarrow |\mathcal{K}^{(i,j)}| + 1$ // Right hand side of (3.5)
$x^* \leftarrow$ Solve LP relaxation of (3.4)-(3.6)
for $i = 1$ **to** $|V|$ **do**
    pick, the kth variable, which is the ith largest component in x^*
    find the set of constraints $C \subseteq E$ including kth variable
    if $y_{ij} > 0, \quad \forall (i,j) \in C$ **then**
        $x_k \leftarrow 1$
        $y_{ij} \leftarrow y_{ij} - 1, \quad \forall (i,j) \in C$
    end if
end for
Return x

If this is the case, then $x_i^* + x_j^* + \sum_{k \in \mathcal{K}^{(i,j)}} x_k^* < 1$ because $|\bar{\mathcal{K}}^{(i,j)}| \leq f$. This contradicts our assumption that x^* is the optimal solution of LP1.

Algorithm 4.5 Primal Rounding Algorithm for the MHC Problem

$x_j = 0, \quad \forall j \in V$
$x^* \leftarrow$ Solve LP relaxation of (3.1)-(3.3)
for all $j \in V$ **do**
    if $x_j^* \geq 1/f$ **then**
        $x_j \leftarrow 1$
    end if
end for
Return x

Dual Rounding for the MHC Problem ($DRMHC$): The algorithm proposed by Hochbaum [59] for the set covering problem is applied to obtain an integral MHC. It uses the optimal solution of the dual problem given by

$$\text{maximize} \quad \sum_{(i,j)\in E} y_{(i,j)}, \tag{4.2}$$

$$\text{subject to} \quad \sum_{(i,j)\in E} y_{(i,j)} + \sum_{(j,i)\in E} y_{(j,i)} + \sum_{j\in\mathcal{K}^{(i,k)}} y_{(i,k)} \leq 1, \qquad j \in V, \tag{4.3}$$

$$y_{(i,j)} \geq 0, \qquad (i,j) \in E, \tag{4.4}$$

where $y_{(i,j)}$ is a dual variable corresponding to the coverage constraint for edge (i,j). The steps of the algorithm are given in Algorithm 4.6. The optimal solution of (4.2)-(4.4) is denoted by y^*. The main idea of the algorithm is to set the primal variable to 1 whenever the corresponding dual constraint is tight.

Algorithm 4.6 Dual Rounding Algorithm for the MHC Problem

```
1: x_j = 0  ∀j ∈ V
2: Solve LP relaxation of (4.2)-(4.4)
3: for all j ∈ V do
4:     if Σ_{(i,j)∈E} y*_{(i,j)} + Σ_{(j,i)∈E} y*_{(j,i)} + Σ_{j∈K^{(i,k)}} y*_{(i,k)} = 1 then
5:         x_j ← 1
6:     end if
7: end for
8: Return x
```

SDP Rounding Algorithm for the MHC Problem ($RSDP$): We implemented a rounding algorithm inspired from another method proposed for the minimum vertex cover problem [51]. This rounding method uses the optimal solution of the SDP relaxation, $\mathbf{v}^*$, and returns the set $S = \{j \in V | \mathbf{v}_0^{*T}\mathbf{v}_j^* > 0\}$ as an approximate solution. This solution is not necessarily a feasible hub cover but the number of uncovered edges is much less with respect to the number of covered edges. Hence, we propose Algorithm 4.7, which obtains S and then repairs the feasibility by iteratively selecting a vertex $i \in V \setminus S$ which covers the highest number of uncovered edges until all edges are covered.

Numerical Experiments. In this section, we conduct a set of experiments to test the performance of the LP and SDP relaxations as well as the rounding algorithms using

Algorithm 4.8 SDP Algorithm for the MHC Problem

1: $x_j = 0 \ \ \forall j \in V$
2: $\mathbf{v}^* \leftarrow$ Solve SDP relaxation of (3.27)-(3.30)
3: **for all** $j \in V$ **do**
4: **if** $\mathbf{v}_0^{*T}\mathbf{v}_j^* > 0$ **then**
5: $x_j \leftarrow 1$
6: **end if**
7: **end for**
8: Find the set of uncovered edges $U \subseteq E$
9: **while** $|U| > 0$ **do**
10: Find the vertex j that covers the maximum number of edges in U
11: $x_j \leftarrow 1$
12: $U = U \setminus (i,k) \ \ \forall (i,k)$ covered by vertex j
13: **end while**
14: **Return** x

the optimal solutions of those relaxations. Here, we list our problem classes and their descriptions. Our data set includes a total of 210 graphs (30 graphs from each class) with known optimal solutions. The first five classes are from a well-known graph database by Santo et al. [93] and the others are synthetically generated graphs used in various application areas. The LP and SDP relaxations are obtained by MATLAB 2010b. To solve the SDP relaxation, we used the SDPA-M solver which is a MATLAB interface for the semidefinite programming algorithm (SDPA) solver developed by Kojima et al. [69]. The solver is developed to solve small and medium size semidefinite programming models. Therefore, our problem set includes only small to medium size instances. The number of vertices and edges range from 20 to 1000.

Rounding algorithms may return a solution, in which some edges are covered several times. Therefore, we applied a postprocessing algorithm to decrease the number of redundant nodes in the hub cover and improve the solution quality. Algorithm 4.9 summarizes the iterations of the postprocessing algorithm. After obtaining the solution by any of the rounding algorithms in line 1, we compute the number of times that each edge is covered by the selected vertices. In line 4, for each vertex in the solution, we check if the vertex is redundant. If it is redundant, then we remove that vertex from the solution and update the number of times each edge is covered by the remaining vertices.

Algorithm 4.9 Postprocessing Algorithm

1: Get the solution x from any one of the rounding algorithms
2: $C_{(i,j)} = x_i + x_j + \sum_{k \in \mathcal{K}^{(i,j)}} x_k \quad \forall (i,j) \in E$
3: $V' = \{j \in V \mid x_j = 1\}$
4: **for all** $j \in V'$ **do**
5: Find the set of edges, E' covered by vertex j
6: **if** $C_{(i,k)} > 1, \quad \forall (i,k) \in E'$ **then**
7: $x_j \leftarrow 0$
8: $C_{(i,k)} \leftarrow C_{(i,k)} - 1$
9: **end if**
10: **end for**

Experimental Results. In this section, we carried out a computational experiment to test the performances of the relaxation models and rounding methods on various types of graph databases. First, we compare the lower bounds obtained by the LP and SDP relaxations over all instances. The empirical cumulative distributions in Figure 4.15 indicate that the LP relaxation gives a tighter lower bound relative to the SDP relaxation. The optimal solutions are denoted by IP in the figure. In almost 90% of the instances, the gaps between the optimal and the LP solutions are less than 10%. On the other hand, the SDP relaxation achieves that gap in 75% of the instances.

Figures 4.16a and 4.16b compare the upper bounds obtained by the rounding methods applied to the optimal solutions of the LP and SDP relaxations before and after postprocessing. The results without postprocessing indicate that the SDP rounding algorithm is superior to the primal and dual rounding algorithms by providing tighter upper bounds. In 70% of the instances, SDP rounding algorithm provides upper bounds with optimality gaps less than 30%. The fraction of the instances decreases to 25% and 15% to obtain the same upper bound by primal and dual rounding algorithms, respectively. On the other hand, surprisingly the rounding algorithm developed for MTS outperforms all other rounding algorithms. The rounding algorithm using the optimal LP solution of MTS provides the optimal solution in almost 45% of the instances. With the postprocessing, the percentage of the instances that can be solved to optimality increases to 55%. The results indicate that postprocessing algorithm eliminates the redundant vertices and improves the solution quality considerably for other algorithms

as well. On the other hand, postprocessing does not change the relative performances of the algorithms. Nonetheless, $PRMHC$ and $DRMHC$ derive the most benefit from postprocessing. The cumulative fraction of the instances for which $PRMHC$ returns optimal solutions changes from 15% to 40% by postprocessing. The corresponding change for $DRMHC$ is from 8% to 32%. After the postprocessing algorithm, in 70% of the instances, the SDP rounding algorithm provides upper bounds with optimality gaps less than 5%. Without postprocessing the same optimality gap is achieved in about 25% of the instances. The fraction of the instances decreases to 45% and 35% to obtain the same upper bound by $PRMHC$ and $DRMHC$ with the postprocessing algorithm.

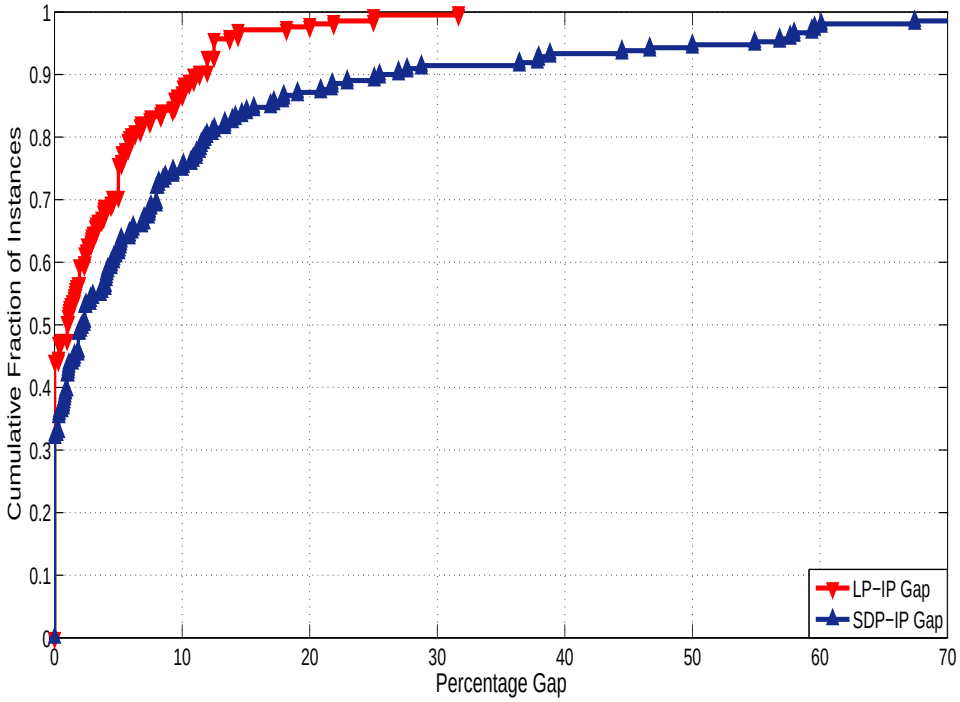
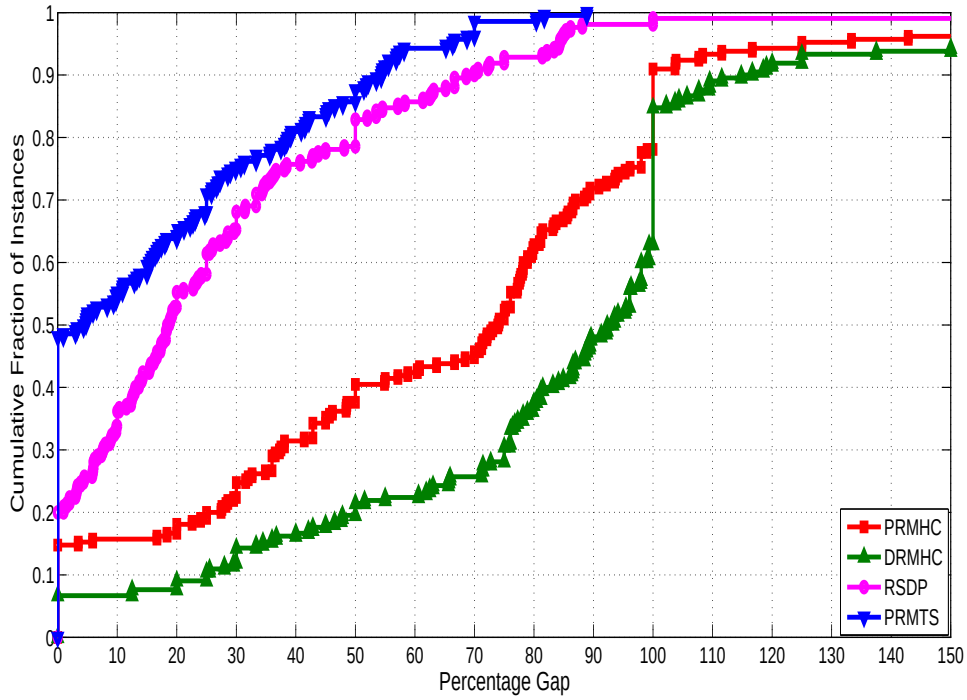
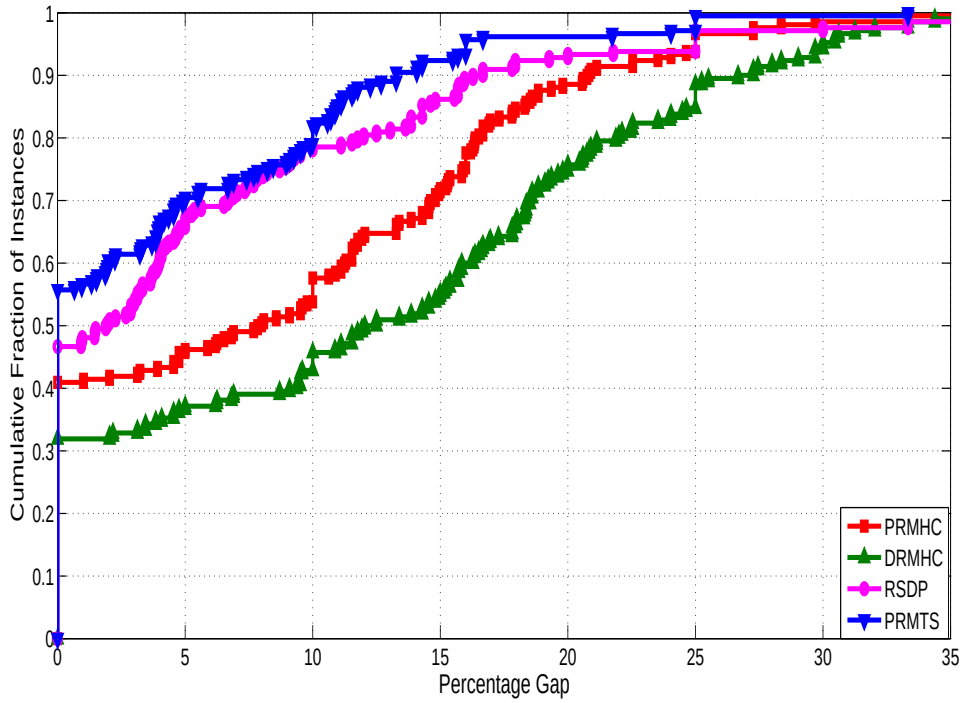


Figure 4.15: The empirical cumulative distributions of the optimality gaps of LP and SDP relaxations

Finally, we analyze the rounding algorithms over problem classes to figure out if the performance of the algorithms changes with respect to the problem classes. Figure 4.17 summarizes and compares the performances of the rounding algorithms over different problem classes. The results indicate that the performances of the algorithms depend on the problem classes. The primal and the dual rounding algorithms applied



(a) Before postprocessing



(b) After postprocessing

Figure 4.16: The empirical cumulative distributions of the optimality gaps of the rounding algorithms before and after postprocessing

to the optimal LP relaxation of MHC are the most sensitive algorithms. On the other hand, the primal rounding algorithm for the complementary problem MTS is the least sensitive algorithm over the problem classes. The variations of the performances of the algorithms are generally low for the mesh graphs in classes (d) and (e).

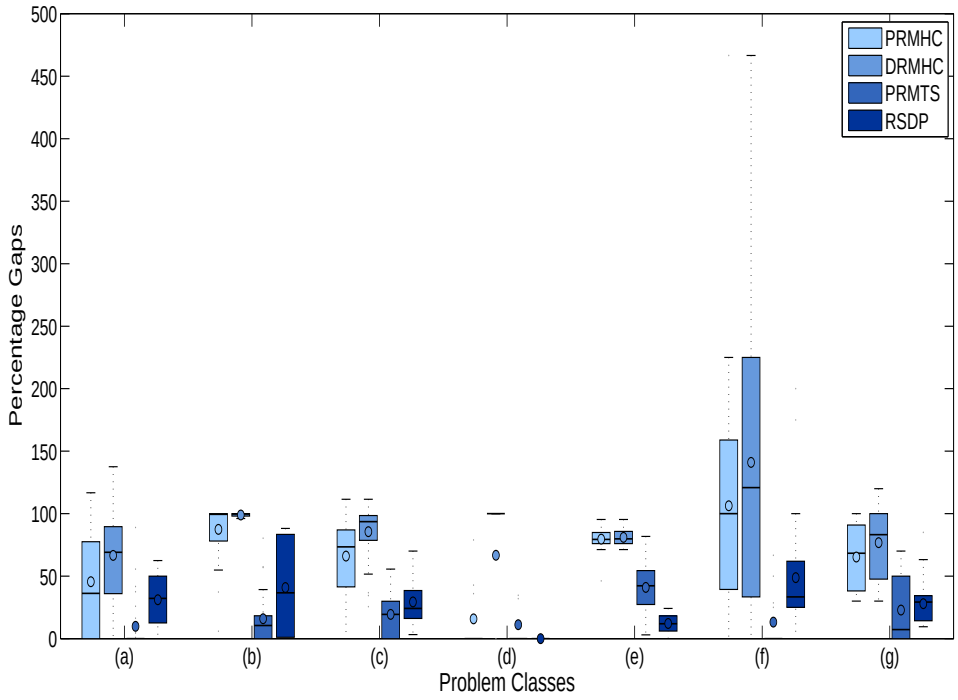


Figure 4.17: The variation of the optimality gaps of the rounding methods with respect to the problem classes.

Chapter 5

APPLICATION: GRAPH QUERY PROCESSING

"Torture the data, and it will confess to anything".

Ronald Coase

Increasing popularity of graph databases has attracted many researchers to focus on graph query processing in recent years. Graph databases keep relational data in various applications such as social networks, Web, protein interactions and so on. MHC has been first introduced as a new graph representation model to expedite graph queries [64]. Managing very large graphs has two main drawbacks: (i) large memory is required to store and process the graphs; (ii) response times to process large graph databases are very high. MHC is proposed as a graph representation model and a query processing strategy [64]. Jamil [64] shows that with this representation model, efficient query processing is possible for generalized undirected graphs without memory limitations. It offers strategic advantages and facilitates construction of candidate graphs from graph fragments.

Query processing, known as graph matching is to find one-to-one mapping between the nodes of a query graph and a database graph under a set of label constraints. Graph matching algorithms look for individual node structures that are identically connected and then these individual matches are combined to see if the composed structure is the target graph. The cost of this search usually is dominated by the cost of piecing together the components and testing if the process is yielding the target graph. We can contemplate several different types of graph matching that can be conceived as

the variants of subgraph isomorphism though in the literature, only the structural isomorphs and match isomorphs defined below are prevalent. MHC is proposed for these two types of matching but by requiring that the two graphs have equal number of nodes, we can also achieve graph isomorphism.

- *structural subgraph isomorph*, where only the node IDs (not the labels) are mapped from query graphs to data graphs using an injective function.
- *label subgraph isomorph*, on the other hand, requires an injective mapping of both node IDs and node labels from query graph to data graph.
- *full subgraph isomorph* extends label subgraph isomorphic matching to include edge labels in the mapping.
- *match subgraph isomorph* uses an equality function on the definition of full subgraph isomorph to achieve exact matching of node and edge labels while maps node IDs using an injective function.

5.1 Minimum Hub Cover: Graph Representation Model

Traditional graph representation $G(V, E)$ does not carry any structural information of which vertices are a part of, and they are not visible until structures are constructed from the set of edges. To ease computational hurdles and aid analysis, some models have used vertices and their neighbours as a unit of representation [98], i.e., $r_v = \langle v, N \rangle$ where v is vertex in V, and N is a set of neighbors such that $(v, n) \in E \Rightarrow n \in N$. A graph is then modeled as a set of such units. While this representation captures some structural cues, it still is pretty basic.

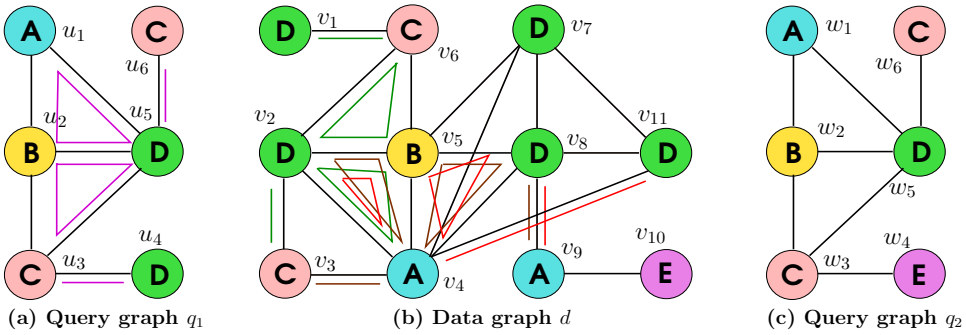
We believe representing a graph as a set of hubs is a prudent compromise because it assures a deterministic model, and yet offers a realistic chance of efficient storage and processing of graph queries. It is deterministic because each hub can be represented as a triple of the form $r_v = \langle v, N_v, B_v \rangle$, called a *graphlet*, where $v \in V$ is a node in graph $G(V, E)$, and $N_v \subseteq V$, and $B_v \subseteq E$ such that for each $v \in V$ all $n \in N_v$ are its immediate neighbors, and every edge $b \in B_v$ are edges involving neighbors in N_v. For

unlabeled and undirected graphs, this representation model is sufficient but for labeled and directed graphs, this simple model can be extended without any structural overhaul. For example, a hub of a node labeled undirected graph can be represented simply as $r_v = \langle v, L_v, N_v, B_v \rangle$ where L_v additionally represents the node label. The hubs in a fully labeled graphs can be modeled as yet another extension as $r_v = \langle v, L_v, N_{v_l}, B_{v_l} \rangle$ where N_{v_l} are a set of pairs (n, n_l) and B_{v_l} are triples(n_1, n_2, e_l) such that n_l and e_l are edge labels for edges between the hub and the neighbors, and among the neighbors respectively. The directionality of the edges can be captured by partitioning the sets N_v and B_v to imply directions. For example, the expression $\langle v, (N_{v_t}, N_{v_f}), (B_{v_t}, B_{v_f}) \rangle$ means that (i) there are edges from v to every node in N_{v_t}, and from N_{v_f} to v, and (ii) for each edge (n_1, n_2) in B_{v_t}, the sink node is n_2, and for edges in B_{v_f}, it is reversed. We can now simply define a graph as a set of graphlets, i.e., $G = \bigcup_{\forall v, v \in V} r_v$.

5.2 Graph Matching

Based on the notion of hub, we are able to reduce the number of query hubs by solving the MHC problem for a query graph. Given the fact that a query graph may have multiple MHCs, we devised a technique to perform graph matching that takes a query graph q, a data graph g, and a set of MHCs M of q as input, and it outputs all of the possible matchings of q in g. After computing the MHCs of a query graph, we are able to use them to perform graph matching. We have devised a technique based on graphlets and MHCs that comprises three steps: 1) Computation of the search space; 2) Computation of a MHC plan; 3) Processing the MHC plan.

In the first step, the goal is to compute the search space, i.e., the nodes of g that match each node of q. If g is either labeled or unlabeled, we take advantage of the representation of the graphs by means of hubs in the following way: let r_u and r_v be two hubs in q and g, respectively; node v belongs to the search space of u if and only if the number of neighbors/triangles of r_u is less or equal than the number of neighbors/triangles of r_v. Thanks to this property, we are able to prune the search space even when we are dealing with unlabeled graphs. Additionally, for labeled graphs, v belongs to the search space of u if and only if their labels match.



(a) Query graph q_1 **(b) Data graph d** **(c) Query graph q_2**

Figure 5.1: Example: Query graphs q_1 and q_2, and data graph d

EXAMPLE 5.1 *Let us determine the search space for the query graph in Figure 5.1(a) with respect to data graph in Figure 5.1(b).*

Graphlet representation of query graph, q_1***:***

$\langle u_4, \{u_3\}, \{\emptyset\}\rangle$

$\langle u_5, \{u_1, u_2, u_3, u_6\}, \{(u_1, u_2), (u_2, u_3)\}$

Graphlet representation of the data graph, d

$\langle v_1, \{v_6\}, \{\emptyset\}\rangle$

$\langle v_2, \{v_3, v_4, v_5, v_6\}, \{(v_3, v_4), (v_4, v_5), (v_5, v_6)\}$

$\langle v_3, \{v_2, v_4\}, \{(v_2, v_4)\}$

$\langle v_4, \{v_2, v_3, v_5, v_7, v_8, v_{11}\}, \{(v_2, v_3), (v_2, v_5), (v_5, v_7), (v_5, v_8), (v_7, v_{11}, (v_8, v_{11}))\}$

$\langle v_5, \{v_2, v_4, v_6, v_7, v_8\}, \{(v_1, v_2), (v_2, v_4), (v_4, v_7), (v_4, v_8), (v_7, v_8)\}$

$\langle v_6, \{v_1, v_2, v_5\}, \{(v_2, v_5)\}$

$\langle v_7, \{v_4, v_5, v_8, v_{11}\}, \{(v_4, v_5), (v_4, v_{11}), (v_5, v_8), (v_8, v_{11}\}$

$\langle v_8, \{v_4, v_5, v_7, v_9, v_{11}\}, \{(v_4, v_5), (v_4, v_7), (v_4, v_{11}), (v_5, v_7), (v_7, v_{11}\}$

$\langle v_9, \{v_8, v_{10}\}, \{\emptyset\}\rangle$

$\langle v_{10}, \{v_9\}, \{\emptyset\}\rangle$

$\langle v_{11}, \{v_4, v_7, v_8\}, \{(v_4, v_7), (v_4, v_8), (v_7, v_8)\}\rangle$

Taking boundaries and neighbours into account, the search space of our example is given as below. Notice that v_6 cannot be a candidate for u_5 because both the number of neighbours and boundary edges of u_5 are greater than that of v_6.

$u_4 : \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}\}$

$u_5 : \{v_2, v_4, v_5, v_7, v_8\}$

The second step consists of selecting a MHC from the input set and ordering it to achieve a good matching performance. It is well-known that different orderings may entail very different computation times when performing the graph matching task [56]. To compute an ordering, we first compute the ordering of all of the nodes of the query graph in a similar way as in [56], in which a search order is computed by analysing the costs of the joins of the query nodes. This search order is also called the query plan, which is represented as a binary tree in which the leaves are query nodes and the internal nodes are join operations. The cost of a join is estimated as the product of the sizes of the joined collections. In the case of a leaf node in the order, the collection size is equivalent to the number of nodes in the search space; an internal node in the order is estimated as the product of the sizes of the collections reduced by a factor.

Our technique first computes a query plan p for the whole query graph and, then, it computes a MHC plan to be as similar to p as possible. For instance, assume that a query plan for query graph q_1 in Figure 5.1(a) is $u_1 \bowtie u_2 \bowtie u_3 \bowtie u_5 \bowtie u_6 \bowtie u_4$. We restrict ourselves to left-deep query plans, in which the outer node of each join is always a leaf node. Note that q_1 has two MHCs: $\{u_4, u_5\}$, and $\{u_3, u_5\}$. Our technique processes a whole query hub at a time, therefore, we compute all possible orderings of the MHCs and their associated whole query orderings, which are the following:

$$u_5 \bowtie u_3 \rightsquigarrow u_5 \bowtie u_1 \bowtie u_2 \bowtie u_3 \bowtie u_6 \bowtie u_4$$

$$u_3 \bowtie u_5 \rightsquigarrow u_3 \bowtie u_2 \bowtie u_1 \bowtie u_6 \bowtie u_5 \bowtie u_4$$

$$u_5 \bowtie u_4 \rightsquigarrow u_5 \bowtie u_1 \bowtie u_2 \bowtie u_3 \bowtie u_6 \bowtie u_4$$

$$u_4 \bowtie u_5 \rightsquigarrow u_4 \bowtie u_3 \bowtie u_2 \bowtie u_1 \bowtie u_6 \bowtie u_5$$

As a conclusion, one of the first or the third MHC plans are preferred since their associated query plans are the most similar ones to the original query plan. [91] also takes into account all MHC solutions if the problem has alternate optimal solutions.

They compute the least cost query plan for each set of optimal solution and select the least query plan among all. The optimal solutions are provided by solving the mathematical programming model given as follows:

$$\text{minimize} \sum_{j \in V} x_j, \tag{5.1}$$

$$\text{subject to } x_i + x_j + \sum_{k \in \mathcal{K}^{(i,j)}} x_k \geq 1, \qquad (i,j) \in E, \tag{5.2}$$

$$\sum_{j \in HC^i} x_j \leq |HC^i| - 1, \qquad i \in \{1, \ldots, t-1\}, \tag{5.3}$$

$$x_j \in \{0,1\}, \qquad j \in V. \tag{5.4}$$

The IP formulation (5.1)-(5.4) is solved to compute the t^{th} optimal solution by adding a set of constraints (5.3) to the IP formulation (3.1)-(3.3). Here, HC^i includes all variables that are set to 1 in the i^{th} optimal solution. Constraints (5.3) ensure that new optimal solution obtained at the t^{th} iteration is different than those obtained at the previous iterations. Algorithm 5.1 iterates as long as the cardinality of the optimal solution is equal to the that of the first optimal solution.

Algorithm 5.1 Computing All Optimal Solutions

```
t ← 1
Solve (5.1)-(5.4)
HC^t = {j ∈ V | x_j = 1}
while |HC^t| == HC^1 do
    Solve (5.1)-(5.4)
    t ← t + 1
    HC^t = {j ∈ V | x_j = 1}
end while
    return {HC^1, ..., HC^{t-1}}
```

In the final step, we use the previously computed search space and MHC plan to perform the graph matching. Our technique takes the initial query hub according to the plan, and it uses the search space to find those data hubs that may match with it.

Then, the graph matching task focuses on the structural unification of a query and a data hub, i.e., we have to match all the neighbors and triangles of the query hub with some of the neighbors and triangles of the data hub. When the whole query hub is matched, we perform a recursive call to process the next query hub in the MHC plan. Note that, in the consequent calls, we are able to ground the query nodes with those values that are already matched. When the whole MHC is processed, we report the complete matching and continue the backtracking process until all matchings are found.

EXAMPLE 5.2 *Suppose we select u_5, u_4 as a hub cover in q_1. The cost of query $u_5 \bowtie u_4$ is less than $u_4 \bowtie u_5$ so we select $u_5 \bowtie u_4$ as a query plan. The hub node u_5 has five candidates as stated in Example 5.0.1. Suppose $u_5 = v_2$, then the graphlet representation of query nodes of q_1 change as follows:*
$\langle u_4, \{u_3\}, \{\emptyset\}\rangle$
$\langle v_2, \{u_1, u_2, u_3, u_6\}, \{(u_1, u_2), (u_2, u_3)\}$

Next, we have to find a one-to-one mapping between the neighbours of u_5 and v_2 by considering the connections among the neighbours i.e. the boundary edges. Suppose $u_1 = v_4$, $u_2 = v_5$, $u_3 = v_6$ and $u_6 = v_3$. After those mappings, graphlet representation is given as follows:
$\langle u_4, \{v_6\}, \{\emptyset\}\rangle$
$\langle v_2, \{v_4, v_5, v_6, v_3\}, \{(v_4, v_5), (v_5, v_6)\}$

Once we map all the vertices in u_5, we select the second hub node u_4. Remember that all the nodes of d are the candidates of u_4. However, previous mappings reduces the search space of u_4. The candidates are the hub nodes which has a neighbour v_6. We select v_1 as a candidate so $u_4 = v_1$. Since we find a one-to-one mapping for all query nodes so we are done. We can continue like that to generate all possible subgraph isomorphs in Figure 5.1 (b), which are represented as different colors.

5.3 Experimental Analysis

To experimentally show that the computation of MHCs is worthy, we design an experiment in which we compare four techniques: the original implementation of GraphQL [72] (GQL in short), an implementation of our technique that randomly selects orderings of all of the query nodes (RND in short), another implementation of our technique that uses query plans including all of the query nodes (PLN in short), and the implementation of our technique that uses MHC plans (MHC in short). Note that the results regarding our graph matching technique that we show in this thesis are just for motivational purposes, a complete description of the graph matching technique is available at [91].

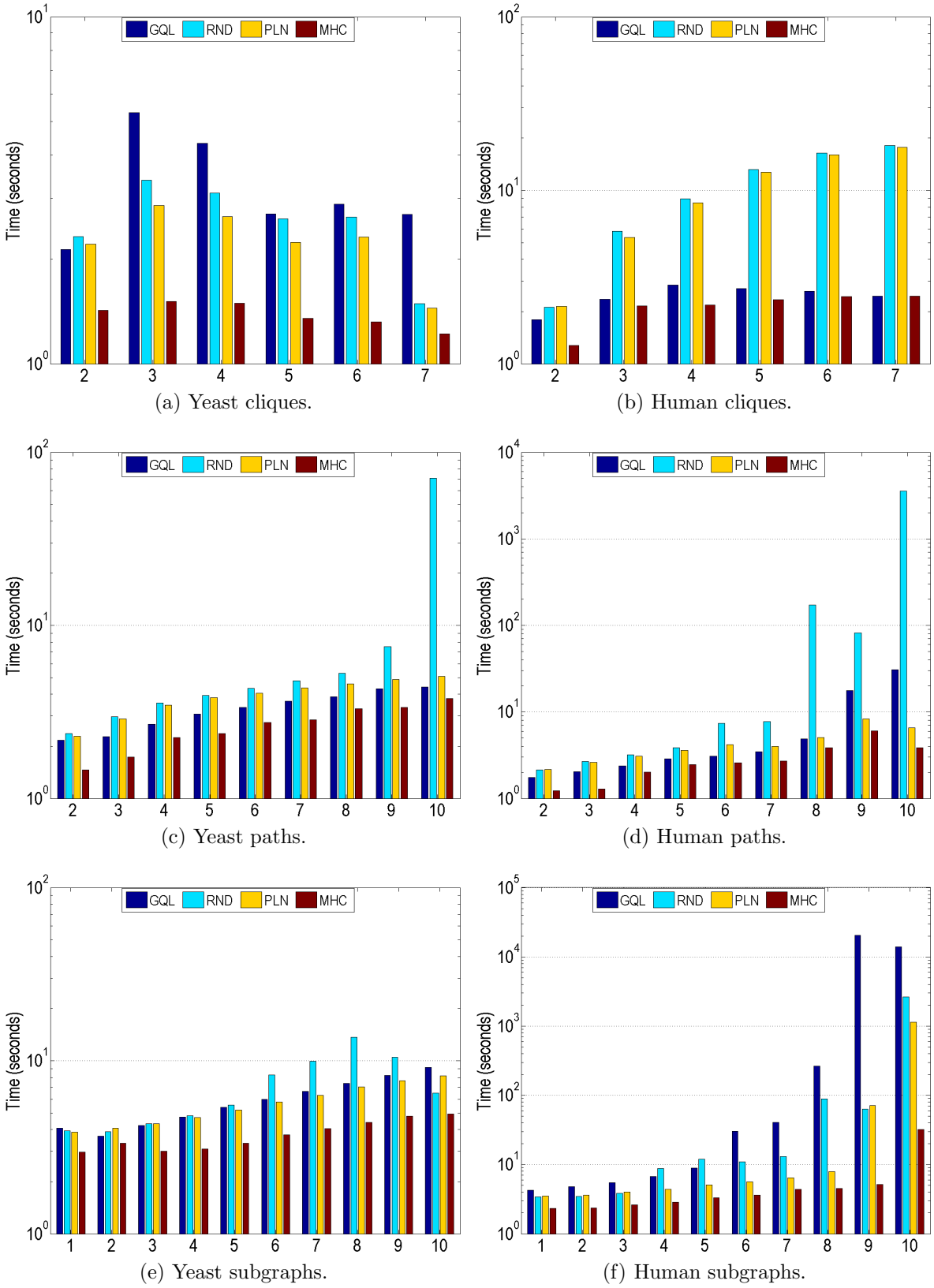
One important issue with respect to RND is that, for moderate large graph queries, the number of node orderings can be huge, e.g., if the query comprises 10 nodes, the possible number of orderings is $10! = 3,628,800$. To reduce this number, we focus on those orderings that ensure connectivity, i.e., they do not perform all possible order of nodes. For instance, for query graph q_1 in Figure 5.1(a), one possible ordering is $u_1 \bowtie u_4 \bowtie u_6 \bowtie u_2 \bowtie u_5 \bowtie u_3$, which is discarded since nodes u_1 and u_4 are adjacent in the order but they are not connected.

To provide a more precise figure on the number of orderings to test in RND, we adapt a statistical technique based on Cochran's formula [90]. In this case, we consider all of the possible orderings of query nodes as a population, then, we rely on Cochran's formula to estimate the sample size, i.e., the number of random orderings to execute. This formula is based on the variance of a target variable and it indicates if the selected sample size statistically represents the whole population, i.e., if the number of tested random orderings is enough to statistically guarantee that their behavior is similar as the whole set of possible orderings. Therefore, we execute a number of random orderings (in our experiments we fixed this number to 20) and measure the times of each order to perform the graph matching task. Then, we use Cochran's formula to estimate the number of orderings that we still have to test and we iterate until the number of tested orderings is greater or equal than the number provided by Cochran's formula. The final

times for a specific query graph is the average of the times of all the random orderings executed.

We used the Yeast and the Human data sets to perform our experiments [72]. In the former, each node represents a unique yeast protein and each edge represents an interaction between two proteins. The latter models a subset of the protein-protein interaction network for homo sapiens. Both Yeast and Human query sets are similar and comprise clique, path and subgraph queries. Clique queries consist of complete query graphs that range from 2 to 7 nodes. Path queries consist of paths connecting a number of nodes, ranging from 2 to 10 nodes. Subgraph queries consist of randomly-generated subgraphs, ranging from 1 to 10 edges. For each type of query (clique, path and subgraph), there exist 1,000 queries with randomly-generated node labels.

Our experiments were run on a computer equipped with a four-threaded Intel Xeon 3.10 GHz CPU and 16 GB RAM, running on Windows 7 Professional (64-bits). Figure 5.2 presents our experimental results, in which the X axis represents the size of the queries, and the Y axis represents the total time in seconds of performing graph matching over the 1,000 queries that a given size comprises. Note that the Y axis is represented in logarithmic scale. As our experimental results show, our MHC technique outperforms the rest of the techniques in all of our experiments and, as a conclusion, it is worthy to compute the MHCs of a query graph to perform graph matching.



(a) Yeast cliques.

(b) Human cliques.

(c) Yeast paths.

(d) Human paths.

(e) Yeast subgraphs.

(f) Human subgraphs.

Figure 5.2: The performance of the graph matching algorithm when a MHC solution is used to obtain a query plan vs other query plans

Chapter 6

CONCLUSION

"A conclusion is simply the place where you got tired of thinking."

Dan Chaon

The minimum hub cover is a new $\mathcal{NP}$-hard optimization problem that has been recently introduced to the literature in the context of graph query processing on large graph databases. In this thesis, we define the problem as an optimization problem and present a standard set covering programming formulation. We also prove that the problem belongs to the class of $\mathcal{NP}$-hard problems.

Similar to the problems in that class, solving the problem to optimality is computationally intractable especially for large instances. Hence, we presented several new mathematical programming formulations along with their relaxations for the minimum hub cover problem. We also introduced two novel rounding algorithms $RSDP$ and $PRMTS$, and compared those with two well-known algorithms proposed for the set covering problem in the literature. The results indicate that the algorithms proposed in this study are superior to the benchmark algorithms in terms of solution quality.

We also analyze the MHC problem on planar graphs. We propose an approximation algorithm to find an approximate solution to the MHC problem. The algorithm decomposes the planar graphs into smaller subgraphs with manageable sizes so it always returns a feasible solution even for the large-scale problems. In the literature, there are some approximation algorithms proposed for the related problems such as the minimum vertex cover and set covering problem. Unlike those algorithms, our ap-

proximation bound is not constant and can be improved by decreasing the number of subproblems to be solved at the expense of an increase in the computation time. We also investigate the empirical performance of the algorithm extensively. Our computational results depict that the empirical performance of the algorithm is far better than its theoretical performance. Alternatively, we propose a decomposition-based heuristic without a proven performance bound. However, it obtains comparable results relative to the approximation algorithm in terms of solution quality. Its solution time is on the average several times less than that of the approximation algorithm. Moreover, we adopt a well-known dynamic programming algorithm to solve the MHC problem on planar graphs.

We employ different algorithms to show the trade-offs between optimality and computation time over different graph types. We conduct computational experiment and analyse solution methods in different axes such as optimal solvability of MHC, computational cost of optimal solution and quality of the solutions. Computational results demonstrate that though computational hard, query processing and optimization using MHC and subgraph isomorphism is computationally feasible and intellectually intriguing.

We discuss that the solution quality of the approximation algorithm is affected by the optimal solution of a subproblem selected among various alternate solutions. Since it is very time consuming to try all combinations of optimal solutions, finding a combination that is good enough for a particular application is an interesting question that we plan to address in our future research. The planar graph decomposition algorithm, by its nature, is amenable to a parallel implementation. In this study, we have used a straightforward shared-memory implementation of the algorithm that helped us save significant computation time. In fact, it is possible with the proposed approach to partition a graph and make use of multiple memory locations in a network. This is of interest to those practitioners, who deal with huge-scale graphs for their problems that are difficult to manage on a single computer. Therefore, obtaining computational results in a distributed computing environment is also in our future research agenda.

Our semidefinite programming relaxation may be used to give an approximation

bound for the minimum hub cover problem. However, at this point it is difficult to give such a result for the minimum hub cover problem. Even for special problem classes, where the number of candidate vertices to cover an edge is less than or equal to three, a formal analysis to obtain an approximation bound seems beyond reach. Nonetheless, based upon our empirical results, we conjecture that our SDP relaxation may achieve an approximation bound less than two for the minimum hub cover problem.

We used the SDPA-M solver, which is developed to solve small to medium size instances limited to 2,000 constraints and 2,000 variables. On the other hand, the parallel version of SDPA referred to as SDPARA, can solve instances with up to a million constraints [42]. As a future study, we plan to employ the parallel implementation of the semidefinite programming solver and test the performance of the SDP relaxation in terms of solution time.

We demonstrate that the cost of query plan changes with respect to the set of vertices coming from different optimal solutions or the order of the hub nodes in an optimal solution. Generating the optimal MHC, which will yield the lest cost query plan is very critical for the performance of the subgraph isomorphism computation. Constraint solvers such as `CPLEX` usually do not offer all optimal solutions. Even so, it is not impractical to generate all optimal solutions of an $\mathcal{NP}$-hard problem. Finding an optimal solution that will yield least computation time query among all optimal solutions is an interesting research problem. We plan to use mathematical programming techniques to solve that problem as a future study.

Appendix A

RELATED OPTIMIZATION PROBLEMS

In this section, we list some problems that are similar to the MHC problem.

Definition A.1 (Set Covering Problem) *Given a collection $\mathcal{S}$ of sets over a finite universe $\mathcal{U}$, a set cover $SC \subseteq \mathcal{S}$ is a sub-collection of these sets, whose union is $\mathcal{U}$. When each set in the collection has an associated cost, then the set covering problem is about finding a set cover SC such that the total cost is minimized. If the cost of coverage is the same for each set, then the problem is called as the unicost set covering problem.*

If an edge corresponds to an item, and a set is defined for each vertex whose elements are the edges covered by that vertex, then the connection between SCP and MHC can easily be established. We continue with the minimum hitting set problem which is equivalent to unicost SCP. The problem definition below implies that there exists a dual relationship between the unicost SCP and the minimum hitting set problem.

Definition A.2 (Minimum Hitting Set Problem) *Given a collection $\mathcal{S}$ of sets over a finite universe $\mathcal{U}$, a hitting set $HS \subseteq \mathcal{U}$ is a set which hits every set of $\mathcal{S}$, i.e. $HS \bigcap S_j \neq \emptyset \ \forall S_j \in \mathcal{S}$. The minimum hitting set problem is to find the hitting set with minimum cardinality.*

Definition A.2 implies that for any instance of SCP, there exists an equivalent instance of the minimum hitting set problem by interchanging the sets $\mathcal{U}$ and $\mathcal{S}$.

Definition A.3 (Minimum Vertex Cover) *For a given graph $G = (V, E)$, a subset of the vertices, $VC \subseteq V$, is a vertex cover of G if for every edge $(i, j) \in E$, either $i \in VC$ or $j \in VC$. MVC is solved to find a vertex cover that has the minimum number of vertices.*

Notice that, in the MHC problem, a vertex can cover the edges that are incident to it as well as the edges between its adjacent neighbors. Clearly, the cardinality of the MHC can be far less than that of the cardinality of the MVC due to the additional non-incident edges covered by those vertices in a triangle. Therefore, for triangle-free graphs, the optimal solutions of MHC and MVC naturally coincide. The optimal MHC and MVC solutions are {a,c,f} and {a,c,g,h,k} for the graph illustrated in Figure 1.3.

The definitions of the SCP, MVC and MHC demonstrate that both the MHC and MVC problems are just special cases of the unicost SCP. MVC problem can be generalized for hypergraphs.

Definition A.4 *Let $H(V, E)$ be a hypergraph with a set of vertices V and a set of hyperedges E. Unlike a graph edge, a hyperedge $e \in E$ can connect any number of vertices, i.e., $e \subseteq V$. Then, a set $S \subseteq V$ is a vertex cover or a hitting set of H if for every edge $e \in E$, $e \cap S \neq \emptyset$ holds.*

MVC in graphs are a special case of MVC in hypergraphs when the number of vertices that connect each edge is two for each edge. Also, SCP and MVC in hypergraphs are equivalent when the item set is defined as the set of hyperedges and the sets are defined as vertex sets connecting each edge.

We continue with an optimization problem known as maximum independent set (MIS), which is complementary to MVC. The formal definition follows.

Definition A.5 (Maximum Independent Set) *For a given graph $G = (V, E)$, $IS \subseteq V$ is an independent set if and only if there is no edge in E between any two nodes in IS. MIS is about finding an independent set IS in G of maximum cardinality.*

It is a known fact that MIS is complementary to MVC. That is, the solution of one gives the solution of the other. Formally, the set of vertices defined by $V \setminus IS$ is

the solution of MVC where IS is the independent set in G. The opposite also holds true. The set of vertices defined by $V \setminus VC$ is the solution of MIS, where VC is the vertex cover in G. By using this information, the optimal solution for MIS whose graph is shown in Figure 1.3 can be easily found as $\{b, d, e, f\}$.

Appendix B

SUPPLEMENTARY TABLES

Table B.1: Average solution times of the benchmark algorithms for random graphs

				MBH	GR1	GR2	LSLP	
Group	$\|V\|$	$\|E\|$	η	**Time**	**Time**	**Time**	**BestTime**	**TotTime**
R1	20	39.8		0.01	0.00	0.00	0.00	0.27
R2	60	339.6		0.16	0.00	0.00	0.01	1.45
R3	100	945.8		3.33	0.00	0.00	0.36	3.82
R4	200	3,786.4	0.1	⋆	0.01	0.01	5.12	15.61
R5	600	34,149.8		◇	0.20	0.21	351.01	626.96
R6	1,000	94,922		⋆	0.86	0.88	801.28	1,871.44
R7	20	26.2		0.01	0.00	0.00	0.00	0.24
R8	60	180.2		0.04	0.00	0.00	0.01	1.17
R9	100	488.2	0.05	0.34	0.00	0.00	0.03	3.00
R10	200	1,950.6		⋆	0.00	0.01	3.51	13.01
R11	600	17,536.2		⋆	0.07	0.07	175.99	622.59
R12	1,000	48,704.2		◇	0.25	0.27	695.8	1,956.81
R13	20	19.6		0.00	0.00	0.00	0.00	0.21
R14	60	70.4		0.01	0.00	0.00	0.00	0.72
R15	100	138.8	0.01	0.01	0.00	0.00	0.00	1.54

Continued on next page...

				MBH	GR1	GR2	LSLP	
Group	$\|V\|$	$\|E\|$	η	**Time**	**Time**	**Time**	**BestTime**	**TotTime**
R16	200	442.4		0.17	0.00	0.00	0.09	6.02
R17	600	3,628.4		$\star$	0.01	0.01	150.55	330.47
R18	1,000	9,991.2		$\star$	0.03	0.03	529.46	1,178.18

$\diamond$: Terminated due to time limit and returned a feasible solution.

$\star$: Terminated without giving a feasible solution.

Table B.2: Average solution times of the benchmark algorithms for bounded graphs

				MBH	**GR1**	**GR2**	**LSLP**	
Group	$\|V\|$	$\|E\|$	**valence**	**Time**	**Time**	**Time**	**BestTime**	**TotTime**
B1	20	30		0.02	0.00	0.00	0.00	0.30
B2	60	90		0.02	0.01	0.00	0.00	0.98
B3	100	150	3	0.03	0.00	0.00	0.01	1.94
B4	200	300		0.05	0.00	0.00	0.54	6.02
B5	600	900		0.17	0.01	0.00	81.37	183.85
B6	1,000	1,500		0.35	0.01	0.00	139.09	493.92
B7	20	60		0.01	0.00	0.00	0.00	0.19
B8	60	180		0.02	0.00	0.00	0.20	0.86
B9	100	300	6	0.01	0.00	0.00	0.45	1.76
B10	200	600		0.02	0.00	0.00	2.05	4.89
B11	600	1,800		0.08	0.01	0.01	40.72	128.08
B12	1,000	3,000		0.15	0.01	0.01	98.19	332.09
B13	20	90		0.01	0.00	0.00	0.00	0.44
B14	60	270		0.02	0.00	0.00	0.00	1.33
B15	100	450	9	0.02	0.00	0.00	0.01	2.52
B16	200	900		0.02	0.01	0.00	0.03	6.99
B17	600	2,700		0.08	0.01	0.01	0.37	182.41
B18	1000	4,500		0.23	0.01	0.01	0.95	473.99

$\diamond$: Terminated due to time limit and returned a feasible solution.

$\star$: Terminated without giving a feasible solution.

Table B.3: Average solution times of the benchmark algorithms for irregular bounded graphs

				MBH	**GR1**	**GR2**	**LSLP**	
Group	$\|V\|$	$\|E\|$	**valence**	**Time**	**Time**	**Time**	**BestTime**	**TotTime**
IB1	20	29.6		0.01	0.00	0.00	0.00	0.28
IB2	60	86.6		0.02	0.00	0.00	0.01	0.92
IB3	100	149.8	3	0.03	0.00	0.00	0.01	1.81
IB4	200	299.8		0.03	0.00	0.00	0.11	5.35
IB5	600	899.6		0.08	0.00	0.00	59.14	157.37
IB6	1,000	1,499.2		0.08	0.00	0.00	157.11	414.82
IB7	20	57.8		0.01	0.00	0.00	0.00	0.23
IB8	60	177.8		0.02	0.00	0.00	0.00	0.92
IB9	100	298.4	6	0.03	0.00	0.00	0.05	1.93
IB10	200	598.4		0.08	0.00	0.00	0.92	5.45
IB11	600	1,797.4		0.11	0.01	0.01	40.98	139.97
IB12	1,000	2,997.2		0.15	0.01	0.01	177.74	358.24
IB13	20	85.4		0.02	0.00	0.00	0.00	0.20
IB14	60	266.6		0.02	0.00	0.00	0.01	1.28
IB15	100	446.6	9	0.04	0.00	0.00	0.02	2.59
IB16	200	895.2		0.10	0.00	0.00	1.21	7.26
IB17	600	2,695.6		0.26	0.01	0.01	55.78	192.64
IB18	1000	4,495.6		0.48	0.02	0.01	137.38	498.20

$\diamond$: Terminated due to time limit and returned a feasible solution.

$\star$: Terminated without giving a feasible solution.

Table B.4: Average solution times of the benchmark algorithms for meshes

				MBH	**GR1**	**GR2**	**LSLP**	
Group	$\|V\|$	$\|E\|$	**dim.**	**Time**	**Time**	**Time**	**BestTime**	**TotTime**
M1	16	24		0.00	0.00	0.00	0.00	0.16
M2	64	112		0.01	0.00	0.00	0.00	0.84
M3	100	180		0.02	0.00	0.00	0.01	1.94
M4	196	364	2D	0.02	0.00	0.00	0.03	6.51
M5	400	760		0.04	0.00	0.00	0.25	24.60
M6	784	1,512		0.08	0.00	0.01	1.12	363.19
M7	1,024	1,984		0.11	0.01	0.01	3.61	614.21
M8	27	54		0.01	0.00	0.00	0.00	0.17
M9	64	144		0.03	0.00	0.00	0.01	0.60
M10	125	300		0.02	0.00	0.00	0.03	1.51
M11	343	882	3D	0.04	0.00	0.00	0.20	18.44
M12	512	1,344		0.05	0.01	0.00	0.35	173.91
M13	729	1,944		0.08	0.00	0.01	1.26	347.72
M14	1,000	2,700		0.10	0.01	0.01	1.55	652.03
M15	16	24		0.01	0.00	0.00	0.00	0.21
M16	81	198		0.02	0.00	0.00	0.01	0.68
M17	256	672	4D	0.03	0.00	0.00	0.05	8.02
M18	625	1,700		1.34	0.00	0.00	2.66	247.04
M19	1,296	3,600		0.22	0.01	0.01	2.11	1,057.55

$\diamond$: Terminated due to time limit and returned a feasible solution.

$\star$: Terminated without giving a feasible solution.

Table B.5: Average solution times of the benchmark algorithms for irregular meshes

				MBH	**GR1**	**GR2**	**LSLP**	
Group	$\|V\|$	$\|E\|$	**dim-ρ**	**Time**	**Time**	**Time**	**BestTime**	**TotTime**
IM1	16	26.6		0.01	0.00	0.00	0.00	0.17
IM2	64	123.4		0.05	0.00	0.00	0.01	1.10
IM3	100	199.6		0.09	0.00	0.00	0.01	2.17
IM4	196	402.4	2D-0.20	0.41	0.00	0.00	0.05	6.56
IM5	400	839.8		2.78	0.00	0.00	1.60	24.46
IM6	784	1,667.8		$\diamond$	0.01	0.00	47.34	354.43
IM7	1,024	2,187.6		$\diamond$	0.01	0.01	66.05	598.2
IM8	16	29.6		0.01	0.00	0.00	0.00	0.21
IM9	64	136		0.06	0.00	0.00	0.00	1.13
IM10	100	218.8		0.06	0.00	0.00	0.02	2.18
IM11	196	440.8	2D-0.40	1.14	0.00	0.00	0.24	6.80
IM12	400	919.2		144.56	0.00	0.00	4.29	25.17
IM13	784	1,823		$\star$	0.01	0.01	121.89	362.58
IM14	1,024	2,392.4		$\star$	0.01	0.01	162.74	612.21
IM15	16	31.8		0.01	0.00	0.00	0.00	0.23
IM16	64	148.6		0.04	0.00	0.00	0.01	1.21
IM17	100	239.2		0.23	0.00	0.00	0.01	2.29
IM18	196	479.8	2D-0.60	2.26	0.00	0.00	0.06	7.06
IM19	400	999		1,928.2	0.00	0.00	7.98	25.91
IM20	784	1,980.6		$\star$	0.01	0.00	151.61	375.07
IM21	1,024	2,596.4		$\star$	0.01	0.01	200.34	631.05
IM22	27	58.6		0.01	0.00	0.00	0.00	0.29
IM23	64	155.6		0.02	0.00	0.00	0.01	0.79
IM24	125	324.6		0.09	0.00	0.00	0.02	2.79
IM25	343	950	3D-0.20	1.60	0.00	0.00	0.12	20.46

Continued on next page...

				MBH	GR1	GR2	LSLP	
Group	$\|V\|$	$\|E\|$	**dim-ρ**	**Time**	**Time**	**Time**	**BestTime**	**TotTime**
IM26	512	1,445		3.83	0.00	0.01	0.63	175.49
IM27	729	2,088.4		17.81	0.01	0.01	1.11	348.27
IM28	1,000	2,899.4		855.1	0.01	0.01	1.91	650.22
IM29	27	63		0.01	0.00	0.00	0.00	0.32
IM30	64	168.4		0.03	0.00	0.00	0.01	1.03
IM31	125	348.2		0.31	0.00	0.00	0.02	3.41
IM32	343	1,018.2	3D-0.40	49.76	0.00	0.00	0.12	21.06
IM33	512	1,547		714.89	0.00	0.00	0.75	179.09
IM34	729	2,233.4		⋆	0.01	0.01	1.14	356.45
IM35	1,000	3,098.4		◇	0.01	0.01	2.73	658.38
IM36	27	68.8		0.02	0.00	0.00	0.00	0.37
IM37	64	180.4		0.04	0.00	0.00	0.00	1.05
IM38	125	373.2		0.44	0.00	0.00	0.02	3.61
IM39	343	1,084.8	3D-0.60	341.82	0.00	0.00	0.13	21.72
IM40	512	1,650		⋆	0.01	0.00	0.74	184.12
IM41	729	2,379.2		⋆	0.01	0.01	7.23	363.35
IM42	1,000	3,297.4		⋆	0.01	0.01	63.9	678.23
IM43	16	26.4		0.01	0.00	0.00	0.00	0.20
IM44	81	213.6		0.03	0.00	0.00	0.01	1.04
IM45	256	722.8	4D-0.20	0.33	0.00	0.00	0.13	11.40
IM46	625	1,824.2		19.98	0.00	0.00	1.68	253.01
IM47	1,296	3,858.8		509.04	0.01	0.01	6.55	1,065.88
IM48	16	29		0.00	0.00	0.00	0.00	0.20
IM49	81	229.4		0.04	0.00	0.00	0.01	1.59
IM50	256	773.2	4D-0.40	1.76	0.00	0.00	0.15	11.89
IM51	625	1949		⋆	0.00	0.01	3.43	261.98
IM52	1,296	4,116.8		◇	0.01	0.01	18.53	1,087.81

Continued on next page...

				MBH	GR1	GR2	LSLP	
Group	$\|V\|$	$\|E\|$	**dim-ρ**	**Time**	**Time**	**Time**	**BestTime**	**TotTime**
IM53	16	31.6		0.01	0.00	0.00	0.00	0.23
IM54	81	244.2		0.12	0.00	0.00	0.02	1.77
IM55	256	823.6	4D-0.60	5.94	0.00	0.00	0.15	12.56
IM56	625	2,073		$\star$	0.01	0.01	4.41	270.53
IM57	296	4,374.6		$\diamond$	0.01	0.01	110.33	1,124.86

$\diamond$: Terminated due to time limit and returned a feasible solution.

$\star$: Terminated without giving a feasible solution.

Table B.6: Average solution times of the benchmark algorithms for scale-free graphs

				MBH	GR1	GR2	LSLP	
Group	$\|V\|$	$\|E\|$	$\alpha-\beta$	**Time**	**Time**	**Time**	**BestTime**	**TotTime**
S1	20	107	1.5-2,000	0.04	0.00	0.00	0.01	0.18
S2	20	113	1.5-4,000	0.01	0.00	0.00	0.01	0.16
S3	20	113.8	1.5-10,000	0.01	0.00	0.00	0.01	0.23
S4	20	49.4	2.5-2,000	0.01	0.00	0.00	0.01	0.24
S5	20	75.8	2.5-4,000	0.01	0.00	0.00	0.01	0.24
S6	20	99	2.5-10,000	0.01	0.00	0.00	0.01	0.17
S7	60	718.8	1.5-6,000	0.24	0.00	0.00	0.07	1.53
S8	60	943.4	1.5-12,000	0.26	0.00	0.00	0.08	1.08
S9	60	1125.2	1.5-30,000	5.81	0.01	0.00	0.21	1.59
S10	60	118.4	2.5-6,000	0.02	0.00	0.00	0.02	0.74
S11	60	146.2	2.5-12,000	0.01	0.00	0.00	0.02	0.78
S12	60	239.6	2.5-30,000	0.00	0.00	0.00	0.02	1.08

Continued on next page...

				MBH	GR1	GR2	LSLP	
Group	$\|V\|$	$\|E\|$	$\alpha - \beta$	**Time**	**Time**	**Time**	**BestTime**	**TotTime**
S13	100	1359.4	1.5-10,000	0.5	0.00	0.01	0.20	5.06
S14	100	1952	1.5-20,000	1.12	0.01	0.01	0.21	5.07
S15	100	2830.6	1.5-50,000	71.47	0.01	0.01	0.48	3.88
S16	100	168.6	2.5-10,000	0.01	0.00	0.00	0.03	1.34
S17	100	237.6	2.5-20,000	0.00	0.00	0.00	0.03	1.58
S18	100	385.6	2.5-50,000	0.01	0.00	0.00	0.04	2.01
S19	200	2629.4	1.5-20,000	150.06	0.01	0.01	5.86	17.67
S20	200	4443.8	1.5-40,000	$\star$	0.02	0.02	11.58	27.83
S21	200	7258.2	1.5-100,000	$\star$	0.03	0.04	11.19	32.00
S22	200	306.2	2.5-20,000	0.00	0.00	0.00	0.08	3.88
S23	200	353	2.5-40,000	0.01	0.00	0.00	0.08	4.02
S24	200	655	2.5-100,000	0.01	0.00	0.00	0.11	5.54
S25	600	7,504.2	1.5-60,000	$\star$	0.03	0.03	159.74	492.39
S26	600	12,447	1.5-120,000	$\star$	0.06	0.07	491.14	773.12
S27	600	23,695.6	1.5-300,000	$\diamond$	0.16	0.17	613.51	1,327.97
S28	600	672.8	2.5-60,000	0.02	0.00	0.00	1.23	98.51
S29	600	918.8	2.5-120,000	0.01	0.00	0.00	1.28	107.24
S30	600	1,216	2.5-300,000	0.02	0.00	0.00	1.34	116.67
S31	1,000	11,889.2	1.5-100,000	$\star$	0.06	0.06	433.82	1,201.1
S32	1,000	19,833.8	1.5-200,000	$\diamond$	0.11	0.12	804.48	1,906.62
S33	1,000	39,834.8	1.5-500,000	$\diamond$	0.32	0.34	1,491.73	3,751.9
S34	1,000	1,061.2	2.5-100,000	0.01	0.00	0.00	3.41	271.12
S35	1,000	1,375.4	2.5-200,000	0.03	0.00	0.00	3.44	278.52
S36	1,000	1,824.8	2.5-500,000	0.03	0.01	0.01	3.61	302.21

$\diamond$: Terminated due to time limit and returned a feasible solution.

$\star$: Terminated without giving a feasible solution.

Table B.7: Percentage gaps obtained by planar graph decomposition-based heuristic for different k values

No	$\|V\|$	$\|E\|$	l	T^{IP}	Gap^{IP}(%)	$Gap^{k=1}$	$Gap^{k=2}$	$Gap^{k=3}$	$Gap^{k=5}$	$Gap^{k=7}$	$Gap^{k=10}$	$Gap^{k=15}$	$Gap^{k=20}$	$Gap^{k=25}$	$Gap^{k=30}$
1	1,000	2,500	100	0.40	0.00	69.86	36.52	23.48	13.62	8.70	6.09	2.90	2.32	1.16	0.87
2	1,000	2,500	100	0.40	0.00	70.06	37.21	23.84	13.95	9.88	4.94	3.20	1.45	1.16	0.87
3	5,000	7,000	100	1.80	0.00	46.62	23.25	15.12	8.37	6.13	4.12	1.22	1.18	0.51	0.31
4	5,000	7,000	100	2.00	0.00	50.12	23.92	16.00	8.47	6.26	3.43	1.97	0.55	0.63	0.00
5	10,000	12,000	150	6.10	0.00	46.70	22.09	15.61	8.56	5.15	3.50	1.82	0.86	0.06	0.43
6	10,000	12,000	150	7.30	0.00	42.38	22.51	15.42	8.67	5.50	3.95	1.66	0.53	0.35	0.04
7	20,000	25,000	200	13.40	0.00	45.87	21.71	14.69	8.62	5.70	4.35	2.25	1.12	0.45	0.30
8	20,000	25,000	200	13.60	0.00	44.10	23.12	15.37	8.06	5.98	4.15	1.83	1.26	0.61	0.51
9	50,000	55,000	500	54.10	0.01	48.22	22.49	14.61	8.60	5.71	4.08	2.46	1.54	0.42	0.57
10	50,000	55,000	500	103.10	0.01	49.39	25.60	16.87	10.27	6.38	4.18	2.88	1.33	0.35	0.33
11	100,000	105,000	1,000	254.70	0.01	39.14	21.75	14.89	9.02	6.31	3.99	2.34	1.22	0.56	0.39
12	100,000	105,000	1,000	360.40	0.01	40.78	23.46	14.73	9.75	6.45	4.31	2.31	1.18	0.76	0.20
13	300,000	306,003	1,500	◇	0.02	41.30	22.11	14.97	9.37	6.83	4.55	2.92	1.26	1.00	0.42
14	300,000	306,002	1,500	◇	0.03	40.33	22.73	15.62	9.19	6.35	4.24	2.87	1.11	0.75	0.60
15	500,000	507,018	2,000	◇	0.12	36.78	21.58	14.94	9.13	6.24	4.46	2.61	1.19	0.73	0.58
16	500,000	507,013	2,000	◇	0.13	35.80	20.98	15.37	9.17	6.79	4.57	2.83	1.19	0.70	0.61
17	800,000	808,461	4,000	◇	0.15	28.69	18.21	13.71	8.90	6.22	4.25	2.30	1.13	0.52	0.39
18	800,000	808,449	4,000	◇	0.15	29.24	17.28	12.58	8.19	6.31	4.29	2.23	1.09	0.57	0.36
19	1,000,000	1,010,574	5,000	◇	0.21	29.46	18.10	13.32	8.54	6.32	4.59	2.58	1.14	0.58	0.45
20	1,000,000	1,010,578	5,000	◇	0.21	30.33	18.01	13.17	8.79	6.39	4.52	2.48	1.27	0.61	0.50

◇: Terminated due to time limit and returned a feasible solution.

⋆: Terminated without giving a feasible solution.

Table B.8: Percentage gaps obtained by approximation algorithm for different k values

No	$\|V\|$	$\|E\|$	l	T^{IP}	Gap^{IP}(%)	$Gap^{k=1}$	$Gap^{k=2}$	$Gap^{k=3}$	$Gap^{k=5}$	$Gap^{k=7}$	$Gap^{k=10}$	$Gap^{k=15}$	$Gap^{k=20}$	$Gap^{k=25}$	$Gap^{k=30}$
1	1,000	2,500	100	0.40	0.00	70.43	36.52	27.25	15.94	8.70	8.41	4.93	3.77	2.03	1.74
2	1,000	2,500	100	0.40	0.00	70.06	38.08	26.74	16.28	10.47	8.72	3.49	4.07	3.78	3.20
3	5,000	7,000	100	1.80	0.00	46.35	22.98	15.44	10.41	6.64	4.75	1.73	2.47	1.65	1.06
4	5,000	7,000	100	2.00	0.00	49.33	23.92	16.27	10.36	6.78	4.26	3.23	2.80	1.73	1.69
5	10,000	12,000	150	6.10	0.00	46.74	22.46	16.78	8.56	7.21	4.13	2.35	2.92	1.19	2.25
6	10,000	12,000	150	7.30	0.00	42.24	23.02	15.42	9.90	6.54	5.19	3.50	2.09	2.21	1.25
7	20,000	25,000	200	13.40	0.00	45.70	21.71	16.19	10.60	6.12	4.98	3.08	1.95	1.63	0.91
8	20,000	25,000	200	13.60	0.00	44.04	23.21	15.72	10.93	6.31	5.40	2.82	2.82	1.59	2.99
9	50,000	55,000	500	54.10	0.01	47.73	22.49	15.67	11.15	8.47	5.66	4.14	2.84	2.13	1.58
10	50,000	55,000	500	103.10	0.01	48.95	26.07	16.90	10.86	7.92	7.10	6.04	2.87	2.34	3.05
11	100,000	105,000	1,000	254.70	0.01	38.97	21.50	14.89	10.15	6.46	4.72	2.93	2.42	2.12	2.11
12	100,000	105,000	1,000	360.40	0.01	40.83	23.50	16.84	10.72	6.96	5.34	3.63	3.42	1.93	1.98
13	300,000	306,003	1,500	◇	0.02	41.05	21.76	16.35	9.76	7.25	4.73	3.65	2.95	2.05	2.26
14	300,000	306,002	1,500	◇	0.03	40.37	22.72	16.41	9.95	7.20	4.18	3.38	3.43	2.18	2.25
15	500,000	507,018	2,000	◇	0.12	36.93	21.61	15.27	9.24	7.52	5.18	3.27	2.72	2.44	2.42
16	500,000	507,013	2,000	◇	0.13	36.01	21.87	15.86	10.15	6.84	4.69	3.46	2.55	2.36	3.06
17	800,000	808,461	4,000	◇	0.15	28.61	18.29	13.63	9.04	6.53	5.02	3.60	2.77	2.10	1.51
18	800,000	808,449	4,000	◇	0.15	29.33	17.29	12.63	8.88	7.06	5.30	3.34	2.49	2.08	1.71
19	1,000,000	1,010,574	5,000	◇	0.21	29.50	17.92	13.36	8.90	6.93	5.03	3.85	2.70	2.24	2.05
20	1,000,000	1,010,578	5,000	◇	0.21	30.53	18.48	13.85	9.43	6.67	5.36	3.46	2.95	2.24	1.65

◇: Terminated due to time limit and returned a feasible solution.

⋆: Terminated without giving a feasible solution.

Table B.9: Computation times obtained by planar graph approximation algorithm for different k values

No	$\|V\|$	$\|E\|$	l	T^{IP}	$T^{k=1}$	$T^{k=2}$	$T^{k=3}$	$T^{k=5}$	$T^{k=7}$	$T^{k=10}$	$T^{k=15}$	$T^{k=20}$	$T^{k=25}$	$T^{k=30}$
1	1,000	2,500	100	0.40	0.20	0.20	0.20	0.30	0.50	1.00	1.20	2.40	3.70	4.40
2	1,000	2,500	100	0.40	0.20	0.10	0.10	0.30	0.40	0.70	1.20	1.90	3.60	4.60
3	5,000	7,000	100	1.80	0.50	0.60	0.80	1.30	1.90	3.20	7.50	11.40	16.80	22.10
4	5,000	7,000	100	2.00	0.40	0.50	0.70	1.30	1.80	3.40	7.90	12.30	17.20	22.40
5	10,000	12,000	150	6.10	0.50	0.80	1.20	2.10	3.60	7.20	14.90	24.60	37.40	47.70
6	10,000	12,000	150	7.30	0.50	0.90	1.20	2.00	3.30	6.40	15.60	27.30	37.30	53.10
7	20,000	25,000	200	13.40	1.00	1.70	2.50	5.50	9.80	16.60	33.40	59.00	108.30	178.00
8	20,000	25,000	200	13.60	1.00	1.70	2.30	5.10	8.50	15.80	36.60	52.40	114.40	191.20
9	50,000	55,000	500	54.10	1.80	3.20	4.40	7.80	15.10	37.10	81.60	149.40	276.30	549.20
10	50,000	55,000	500	103.10	1.80	3.20	4.30	8.40	16.10	38.20	84.70	157.40	308.20	593.20
11	100,000	105,000	1,000	254.70	3.20	5.50	7.50	13.70	20.70	52.30	144.30	322.60	627.70	1,219.10
12	100,000	105,000	1,000	360.40	3.10	5.30	7.70	13.90	21.60	55.70	153.90	315.00	638.40	1,221.20
13	300,000	306,003	1,500	◇	9.20	15.00	21.80	39.90	70.00	252.60	924.30	1,459.90	2,247.70	4,040.60
14	300,000	306,002	1,500	◇	9.50	14.80	21.60	39.50	71.00	272.30	961.00	1,530.10	2,241.70	3,889.30
15	500,000	507,018	2,000	◇	16.70	26.10	36.70	66.50	153.00	614.90	1,607.00	2,929.30	6,522.60	11,629.10
16	500,000	507,013	2,000	◇	16.80	26.10	37.30	68.20	164.30	627.10	1,693.40	2,971.50	6,663.40	11,769.70
17	800,000	808,461	4,000	◇	36.70	48.10	59.90	102.40	190.50	655.10	2,306.80	4,478.70	6,607.80	11,431.40
18	800,000	808,449	4,000	◇	36.10	48.50	59.80	106.10	212.20	682.90	2,405.30	4,707.70	6,778.00	11,747.00
19	1,000,000	1,010,574	5,000	◇	53.40	67.50	81.50	138.60	267.60	853.10	3,173.20	6,068.60	8,688.80	15,046.50
20	1,000,000	1,010,578	5,000	◇	54.00	66.90	81.30	138.00	267.20	835.60	3,176.90	6,078.10	8,579.80	14,477.20

◇: Terminated due to time limit and returned a feasible solution.

⋆: Terminated without giving a feasible solution.

Table B.10: Computation times obtained by planar graph decomposition-based heuristic for different k values

No	$\|V\|$	$\|E\|$	l	T^{IP}	$T^{k=1}$	$T^{k=2}$	$T^{k=3}$	$T^{k=5}$	$T^{k=7}$	$T^{k=10}$	$T^{k=15}$	$T^{k=20}$	$T^{k=25}$	$T^{k=30}$
1	1,000	2,500	100	0.40	0.20	0.10	0.10	0.20	0.10	0.10	0.20	0.20	0.20	0.10
2	1,000	2,500	100	0.40	0.20	0.10	0.10	0.10	0.10	0.10	0.20	0.10	0.20	0.20
3	5,000	7,000	100	1.80	0.40	0.40	0.50	0.50	0.50	0.40	0.40	0.50	0.50	0.50
4	5,000	7,000	100	2.00	0.30	0.40	0.50	0.60	0.40	0.60	0.40	0.50	0.50	0.70
5	10,000	12,000	150	6.10	0.50	0.60	0.60	0.80	0.90	0.90	1.00	1.00	1.20	1.40
6	10,000	12,000	150	7.30	0.60	0.50	0.60	0.80	0.80	1.00	1.00	1.00	1.20	1.10
7	20,000	25,000	200	13.40	1.10	1.00	1.10	1.70	2.00	2.40	2.10	2.30	2.80	2.60
8	20,000	25,000	200	13.60	0.90	1.10	1.10	1.40	1.90	2.50	2.30	2.60	3.00	2.50
9	50,000	55,000	500	54.10	1.80	1.80	1.90	2.10	2.50	4.30	5.40	9.20	10.50	9.20
10	50,000	55,000	500	103.10	1.80	1.70	1.80	2.00	2.70	4.20	6.00	8.90	9.80	9.50
11	100,000	105,000	1,000	254.70	3.20	2.90	2.80	3.10	3.30	5.80	8.40	16.30	16.20	20.20
12	100,000	105,000	1,000	360.40	3.20	2.90	2.70	3.40	3.50	5.80	8.90	15.80	18.50	19.10
13	300,000	306,003	1,500	◇	9.20	7.50	7.30	8.80	10.60	25.90	40.10	50.70	50.30	67.60
14	300,000	306,002	1,500	◇	9.30	7.40	7.30	8.20	9.80	23.60	40.70	47.90	50.80	83.20
15	500,000	507,018	2,000	◇	16.70	12.70	12.60	14.00	22.20	47.20	80.40	105.90	135.30	212.60
16	500,000	507,013	2,000	◇	16.60	12.80	12.80	13.70	29.30	45.50	87.90	95.90	158.70	208.90
17	800,000	808,461	4,000	◇	36.00	24.80	20.50	22.00	26.40	54.90	88.60	126.70	167.90	204.00
18	800,000	808,449	4,000	◇	35.90	25.30	20.30	22.30	25.60	55.70	90.40	144.50	181.70	219.10
19	1,000,000	1,010,574	5,000	◇	53.70	34.80	27.60	27.80	32.80	70.40	123.50	166.80	232.20	287.50
20	1,000,000	1,010,578	5,000	◇	53.40	34.50	27.60	28.80	32.50	69.60	107.70	168.10	217.60	284.50

◇: Terminated due to time limit and returned a feasible solution.

⋆: Terminated without giving a feasible solution.

Table B.11: The performances of the rounding algorithms on random graphs

Instances			Primal Rounding			Dual Rounding		SDP Rounding			MTS	
NoOfNode	NoOfEdge	IPOPT	LPOPT	IPRound	IPPost	IPRound	IPPost	OFVSDP	SDPRound	IPPost	IPRound	IPPost
20	38	8	8.0	11	8	14	9	7.4	12	8	8	8
20	39	10	10.0	15	11	20	12	10	12	10	10	10
20	40	8	8.0	8	8	8	8	8	12	8	8	8
20	40	8	8.0	10	8	10	8	7.4	8	8	8	8
20	42	9	8.5	16	10	17	10	8	9	9	10	9
60	344	24	22.6	52	29	54	29	17.3	38	30	33	25
60	342	27	25.3	54	30	58	30	22.4	40	31	51	30
60	337	27	25.2	55	32	55	32	22.3	38	30	34	30
60	332	27	24.4	55	31	59	32	20.1	35	29	42	29
60	343	24	22.9	50	28	57	30	17.1	36	28	34	28
20	20	8	8.0	8	8	9	8	8	8	8	8	8
20	19	8	8.0	8	8	13	8	8	10	8	8	8
20	20	8	8.0	8	8	10	8	8	9	8	8	8
20	19	8	8.0	8	8	12	8	8	12	8	8	8
20	20	8	8.0	8	8	9	8	8	12	8	8	8
60	69	25	25.0	33	25	35	25	25	34	28	25	25
60	70	26	26.0	26	26	39	27	26	35	26	26	26
60	73	29	28.5	30	29	51	30	28.7	32	29	29	29
60	72	25	25.0	25	25	34	25	25	38	26	25	25
60	68	25	25.0	31	25	32	25	25	27	26	25	25
100	143	46	46.0	59	46	68	47	46	54	46	46	46
100	139	44	44.0	57	44	57	44	44	60	45	44	44
100	138	46	45.5	67	49	75	50	45.8	69	50	48	46
100	138	43	43.0	58	43	63	43	43	66	45	43	43
100	136	48	48.0	82	51	91	51	48	78	54	48	48
200	439	105	99.5	184	122	198	126	104.2	118	112	106	106
200	441	104	98.0	180	121	194	123	102.4	122	107	121	112
200	447	103	99.0	183	118	195	129	102.3	112	104	103	103
200	436	104	100.0	179	122	200	127	103.3	130	109	109	106
200	449	107	99.0	190	129	197	129	104.5	132	108	114	109

Table B.12: The performances of the rounding algorithms on bounded graphs

Instances			Primal Rounding			Dual Rounding		SDP Rounding			MTS	
NoOfNode	NoOfEdge	IPOPT	LPOPT	IPRound	IPPost	IPRound	IPPost	OFVSDP	SDPRound	IPPost	IPRound	IPPost
100	150	51	50	54	51	100	59	50.05	94	59	92	59
100	150	51	50	81	57	100	57	50.05	86	55	80	57
100	150	51	50	70	55	100	59	50.05	88	57	58	53
100	150	51	50	82	57	100	59	50.05	87	59	71	55
100	150	51	50	79	53	100	61	50.05	85	59	56	51
200	300	101	100	200	115	200	115	100.02	174	115	107	103
200	300	101	100	200	117	200	117	100.02	186	115	140	105
200	300	101	100	200	121	200	121	100.02	184	123	159	111
200	300	101	100	200	119	200	119	100.02	190	119	114	101
200	300	101	100	200	117	200	117	100.02	188	115	112	103
600	900	301	300	599	357	600	357	300.01	556	355	326	305
600	900	301	300	536	347	600	355	300.01	555	355	356	311
600	900	301	300	507	333	600	351	300.01	552	349	368	313
600	900	301	300	583	355	600	357	300.01	546	359	341	303
600	900	301	300	524	335	600	353	300.01	558	345	352	313
60	270	30	30	60	30	60	30	30	31	30	30	30
60	270	30	30	60	30	60	30	30	30	30	30	30
60	270	30	30	60	30	60	30	30	32	30	30	30
60	270	30	30	60	30	60	30	30	30	30	30	30
60	270	30	30	60	30	60	30	30	30	30	30	30
100	450	50	50	100	50	100	50	50	50	50	50	50
100	450	50	50	100	50	100	50	50	53	50	55	50
100	450	50	50	100	50	100	50	50	50	50	50	50
100	450	50	50	100	50	100	50	50	53	50	50	50
100	450	50	50	100	50	100	50	50	50	50	50	50
200	900	100	100	200	100	200	100	100	102	100	129	100
200	900	100	100	200	100	200	100	100	103	100	115	100
200	900	100	100	200	100	200	100	100	101	100	100	100
200	900	100	100	200	100	200	100	100	102	100	100	100
200	900	100	100	200	100	200	100	100	101	100	118	100

Table B.13: The performances of the rounding algorithms on irregular bounded graphs

Instances			Primal Rounding			Dual Rounding		SDP Rounding			MTS	
NoOfNode	NoOfEdge	IPOPT	LPOPT	IPRound	IPPost	IPRound	IPPost	OFVSDP	SDPRound	IPPost	IPRound	IPPost
60	90	31	30	60	37	60	37	30.3	36	32	31	31
60	89	30	30	30	30	60	39	30	35	31	30	30
60	89	30	30	51	33	60	38	30	41	31	38	32
60	90	29	29	37	29	39	30	29	39	31	29	29
60	90	31	30	38	32	60	40	30.7	32	31	32	32
100	150	52	50	71	58	100	60	51.3	62	52	66	57
100	150	51	50	65	54	100	65	50.4	61	53	59	53
100	149	49	49	88	57	97	58	48.8	57	51	49	49
100	150	49	49	65	50	67	51	49	58	51	49	49
100	150	51	49.5	99	63	99	63	50.2	54	52	56	52
200	299	98	98	116	99	123	100	98	123	100	98	98
200	300	101	100	162	112	200	119	100.2	124	105	106	101
200	300	100	99.5	145	108	199	124	99.8	113	103	100	100
200	300	99	99	140	107	175	114	99	113	104	99	99
200	300	97	97	144	103	147	103	97	103	97	97	97
60	269	26	25.3	53	29	54	29	22.9	28	27	30	26
60	267	26	25.3	54	29	54	29	22.3	31	28	34	27
60	264	26	25.7	51	30	52	30	23.9	30	28	36	29
60	265	27	26	51	31	53	31	24.5	34	28	41	28
60	268	26	25.6	55	29	55	29	23.4	36	29	32	27
100	445	43	42.5	74	48	85	50	39.8	66	47	53	45
100	445	44	42.5	72	51	73	53	39.9	68	51	51	46
100	447	43	42.3	79	50	80	50	39.6	70	50	63	47
100	447	46	44.5	86	53	88	53	43.3	59	49	71	51
100	449	44	42.7	76	47	76	47	40.7	71	50	54	45
200	897	89	87.8	155	102	157	103	82.9	147	103	112	91
200	897	89	88	161	99	172	106	84.4	127	95	111	94
200	893	92	91	163	101	172	102	89.4	121	104	125	95
200	897	90	88.3	167	105	182	109	84.7	153	104	140	96
200	892	87	86.7	155	93	157	93	84.4	118	94	113	93

Table B.14: The performances of the rounding algorithms on regular meshes

Instances			Primal Rounding			Dual Rounding		SDP Rounding			MTS	
NoOfNode	NoOfEdge	IPOPT	LPOPT	IPRound	IPPost	IPRound	IPPost	OFVSDP	SDPRound	IPPost	IPRound	IPPost
100	180	50	50	68	58	100	59	50	50	50	66	58
100	180	50	50	68	58	100	64	50	50	50	66	58
100	180	50	50	68	58	100	65	50	50	50	66	58
100	180	50	50	68	58	100	59	50	50	50	66	58
100	180	50	50	68	58	100	61	50	50	50	66	58
196	364	98	98	140	111	196	123	98	98	98	132	111
196	364	98	98	140	111	196	128	98	98	98	132	111
196	364	98	98	140	112	196	122	98	98	98	132	112
196	364	98	98	140	113	196	116	98	98	98	132	112
196	364	98	98	140	111	196	120	98	98	98	132	111
64	144	32	32	32	32	64	39	32	32	32	32	32
64	144	32	32	32	32	64	40	32	32	32	32	32
64	144	32	32	32	32	64	43	32	32	32	32	32
64	144	32	32	32	32	64	35	32	32	32	32	32
64	144	32	32	32	32	64	42	32	32	32	32	32
125	300	62	62	62	62	62	62	62	62	62	62	62
125	300	62	62	62	62	62	62	62	62	62	62	62
125	300	62	62	62	62	62	62	62	62	62	62	62
125	300	62	62	62	62	62	62	62	62	62	62	62
125	300	62	62	62	62	62	62	62	62	62	62	62
81	198	39	39	39	39	39	39	39	39	39	39	39
81	198	39	39	39	39	39	39	39	39	39	39	39
81	198	39	39	39	39	39	39	39	39	39	39	39
81	198	39	39	39	39	39	39	39	39	39	39	39
81	198	39	39	39	39	39	39	39	39	39	39	39
256	672	128	128	229	160	256	169	128	128	128	128	128
256	672	128	128	128	128	256	166	128	128	128	128	128
256	672	128	128	128	128	256	166	128	128	128	128	128
256	672	128	128	128	128	256	167	128	128	128	128	128
256	672	128	128	128	128	256	167	128	128	128	128	128

Table B.15: The performances of the rounding algorithms on irregular meshes

Instances			Primal Rounding			Dual Rounding		SDP Rounding			MTS	
NoOfNode	NoOfEdge	IPOPT	LPOPT	IPRound	IPPost	IPRound	IPPost	OFVSDP	SDPRound	IPPost	IPRound	IPPost
196	403	102	98	196	123	196	123	101.1	102	102	119	107
196	402	107	98	196	123	196	123	102.1	126	111	168	119
196	402	106	98	196	128	196	128	101.8	114	108	150	119
196	402	104	98	152	112	196	124	101.9	108	104	173	115
196	403	104	98	196	129	196	129	101.7	116	107	189	129
196	441	109	98	196	129	196	129	103.9	134	119	131	115
196	441	108	98	196	121	196	121	103.5	122	113	144	118
196	441	109	97.5	195	127	195	127	103.5	129	117	155	115
196	441	109	98	196	122	196	122	104.2	117	110	131	114
196	440	110	98	196	126	196	126	104.7	130	114	140	115
196	481	112	98	196	128	196	128	106.2	122	115	173	125
196	481	110	97.5	195	128	195	128	105.5	129	114	174	122
196	478	112	98	196	131	196	131	106.4	133	118	163	124
196	481	112	98	196	127	196	127	106.1	139	122	189	124
196	478	110	98	196	126	196	126	105.6	136	121	140	121
125	325	64	62.5	125	83	125	83	63.9	66	64	71	65
125	324	67	62.5	125	86	125	86	65.8	70	68	88	73
125	325	66	62.5	125	84	125	84	65.3	72	66	80	67
125	325	66	62.5	125	84	125	84	65.6	67	66	93	72
125	324	67	62.5	125	81	125	81	66,3	71	67	69	67
125	349	69	62.5	125	82	125	82	67.7	73	69	81	72
125	346	69	62.5	125	82	125	82	67.3	72	70	114	84
125	348	71	62.5	125	83	125	83	68.1	78	71	103	82
125	348	69	62.5	125	86	125	86	67.1	82	70	105	76
125	350	71	62.5	125	87	125	87	68.4	78	71	110	81
125	374	73	62,5	125	86	125	86	68.6	83	79	106	79
125	374	73	62.5	125	80	125	80	68.5	82	76	99	81
125	374	71	62.5	125	86	125	86	68.2	86	74	109	81
125	373	71	62.5	125	84	125	84	68.4	82	76	105	77
125	371	71	62.5	125	87	125	87	68.1	80	75	101	80

Table B.16: The performances of the rounding algorithms on scale-free graphs

Instances			Primal Rounding			Dual Rounding		SDP Rounding			MTS	
NoOfNode	NoOfEdge	IPOPT	LPOPT	IPRound	IPPost	IPRound	IPPost	OFVSDP	SDPRound	IPPost	IPRound	IPPost
20	113	4	3.5	8	6	9	6	1.7	5	4	4	4
20	116	4	3.1	9	4	10	4	1.6	4	4	4	4
20	86	4	4	5	4	5	4	3.4	6	4	4	4
20	112	4	3.5	13	5	15	5	1.7	11	5	6	5
20	108	3	3	7	4	13	4	1.7	6	3	3	3
20	113	3	3	3	3	3	3	1.67	5	4	3	3
20	103	4	3.8	9	5	11	5	2.5	7	6	4	4
20	124	4	2.7	12	5	15	5	0.8	5	5	5	5
20	110	4	4	12	5	13	5	2.5	6	6	5	5
20	115	4	3.2	12	5	14	6	1.6	5	5	5	5
20	116	4	3	11	4	17	4	0.9	5	5	5	4
20	108	3	3	3	3	3	3	2.4	4	3	3	3
20	113	4	3.3	12	4	13	5	1.6	7	5	5	4
20	124	3	2.7	17	4	17	4	0.8	4	3	5	4
20	108	4	3.6	8	4	12	4	2.4	6	5	4	4
20	57	6	6	11	6	13	6	4.6	10	8	6	6
20	37	7	7	9	7	10	7	6.5	9	8	7	7
20	43	7	7	11	7	12	7	6.3	8	7	7	7
20	53	7	7	17	7	18	7	5.7	11	7	9	7
20	57	6	6	7	6	8	6	5.5	8	6	6	6
20	77	5	5	6	5	6	5	3.7	6	6	5	5
20	65	5	5	5	5	5	5	3.9	6	5	5	5
20	82	5	5	6	5	6	5	3.7	6	5	5	5
20	68	6	6	9	6	9	6	4.9	8	7	6	6
20	87	6	5.2	12	7	12	7	3.7	7	6	9	6
20	114	4	3.8	11	4	16	5	1.3	12	6	6	5
20	104	4	3.5	6	4	7	4	1.8	5	4	5	4
20	106	4	3	6	4	6	4	1.7	4	4	4	4
20	99	4	3.7	8	4	9	4	2	8	4	4	4
20	72	5	5	6	5	6	5	3.9	6	5	5	5

Table B.17: The performances of the rounding algorithms on planar graphs

Instances			Primal Rounding			Dual Rounding		SDP Rounding			MTS	
NoOfNode	NoOfEdge	IPOPT	LPOPT	IPRound	IPPost	IPRound	IPPost	OFVSDP	SDPRound	IPPost	IPRound	IPPost
20	30	10	9.5	13	10	13	10	9.8	13	10	10	10
20	30	10	9.5	13	10	13	10	9.8	13	10	10	10
20	30	10	9.5	13	10	13	10	9.8	13	10	10	10
20	30	10	9.5	13	10	13	10	9.8	13	10	10	10
20	30	10	9.5	13	10	13	10	9.8	13	10	10	10
20	30	10	9.5	20	11	20	11	9.2	11	10	17	11
20	30	10	9.5	20	11	20	11	9.2	11	10	17	11
20	30	10	9.5	20	11	20	11	9.2	11	10	17	11
20	30	10	9.5	20	11	20	11	9.2	11	10	17	11
20	30	10	9.5	20	11	20	11	9.2	11	10	17	11
50	100	21	20.5	29	22	31	23	19.2	29	21	21	21
50	100	19	19	25	19	27	19	16.9	31	20	19	19
50	100	20	19.5	29	20	29	20	17.6	26	21	23	20
50	100	21	20.7	29	22	34	23	17.8	23	22	22	21
50	100	21	21	35	21	46	21	18.2	27	23	28	21
50	100	20	19.7	40	23	44	23	16.4	29	21	24	21
50	100	21	20.5	39	22	44	22	18.4	25	24	21	21
50	100	21	20.5	42	23	44	23	17.9	30	24	29	23
50	100	20	20	31	21	31	21	17.8	27	21	21	20
50	100	23	22.3	40	23	47	23	20.3	29	24	32	23
50	100	20	20	34	22	36	22	18.5	37	26	23	20
50	100	22	21	39	23	44	23	19.1	26	24	22	22
50	100	22	21	42	23	43	23	18.6	26	23	34	22
50	100	22	20.8	41	24	41	24	19.2	25	22	33	22
50	100	21	21	36	23	36	23	19.3	24	23	21	21
80	150	35	34	63	38	72	40	31.2	47	40	53	35
80	150	32	32	48	32	67	36	28.3	46	33	35	32
80	150	32	31.5	48	33	53	33	28.4	37	33	32	32
80	150	33	33	49	33	53	33	29.2	42	34	33	33
80	150	35	35	52	35	58	35	33.4	46	36	35	35

Bibliography

[1] M. A. Abdulrahim and M. Misra. A graph isomorphism algorithm for object recognition. *Pattern Analysis and Applications*, 1:189–201, 1998.

[2] U. Aickelin. An indirect genetic algorithm for set covering problems. *Journal of the Operational Research Society*, 53(10):1118–1126, 2002.

[3] S. Arora, B. Bollobas, L. Lovasz, and I. Tourlakis. Proving integrality gaps without knowing the linear program. *Theory of Computing*, 2:19–51, 2006.

[4] E. Asgeirsson and C. Stein. Vertex cover approximations: Experiments and observations. *Lecture Notes in Computer Science*, 3503:545–557, 2005.

[5] E. Asgeirsson and C. Stein. Vertex cover approximations on random graphs. *Lecture Notes in Computer Science*, 4525:285–296, 2005.

[6] Z. N. Azimi, P. Toth, and L. Galli. An electromagnetism metaheuristic for the unicost set covering problem. *European Journal of Operational Research*, 205:290–300, 2010.

[7] B. S. Baker. Approximation algorithms for $\mathcal{NP}$-complete problems. *Journal of the Association for Computing Machinery*, 41:153–180, 1994.

[8] S. Balaji, V. Swaminathan, and K. Kannan. Optimization of unweighted minimum vertex cover. *World Academy of Science, Engineering and Technology*, 67:508–513, 2010.

[9] E. Balas and M. C. Carrera. A dynamic subgradient-based branch-and-bound procedure for set covering. *Operations Research*, 44(6):875–890, 1996.

[10] S. Baloch and H. Krim. Object recognition through topo-geometric shape models using error-tolerant subgraph isomorphisms. *IEEE Transactions on Image Processing*, 19:1191–1200, 2010.

[11] R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2:198–203, 1981.

[12] R. Bar-Yehuda and S. Even. On approximating a vertex cover for planar graphs. In *14th ACM Symposium on Theory of Computing*, pages 303–309, San Francisco, California, 1982.

[13] R. Bar-Yehuda and D. Rawitz. On the equivalence between the primal-dual schema and the local ratio technique. *SIAM Journal on Discrete Mathematics*, 19:762–797, 2005.

[14] A. L. Barabasi, Z. Dezso, E. Ravasz, S.H. Yook, and Z. Oltvai. Scale-free and hierarchical structures in complex networks. In *AIP Conference Proceedings*, pages 1–16, Granada, Spain, 2003.

[15] V. Batagelj and U. Brandes. Efficient generation of large random networks. *Phys. Rev. E.*, 71:1–5, 2005.

[16] J. E. Beasley. An algorithm for set covering problem. *European Journal of Operational Research*, 31:85–93, 1987.

[17] J. E. Beasley. A Lagrangian heuristic for set covering problems. *Naval Research Logistics*, 37:151–164, 1990.

[18] J. E. Beasley and P. C. Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94:392–404, 1996.

[19] J. E. Beasley and K. Jornsten. Enhancing an algorithm for set covering problems. *European Journal of Operational Research*, 58:293–300, 1992.

[20] D. Bertsimas and R. Vohra. Rounding algorithms for covering problems. *Mathematical Programming*, 80:63–89, 1998.

[21] D. Bienstock and C.L.Monma. On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmica*, 5:93–109, 1990.

[22] H. Brönnimann and M.T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete and Computational Geometry*, 14:463–479, 1995.

[23] M. J. Brusco, L. W. Jacobs, and G. M. Thompson. A morphing procedure to supplement a simulated annealing heuristic for cost- and coverage-correlated set-covering problems. *Annals of Operations Research*, 86:611–627, 1999.

[24] A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. *Operations Research*, 47(5):730–743, 1999.

[25] A. Caprara, P. Toth, and M. Fischetti. Algorithms for the set covering problem. *Annals of Operations Research*, 98:353–371, 2000.

[26] J. Cardinal, M. Karpinski, R. Schmied, and C. Viehmann. Approximating vertex cover in dense hypergraphs. *Journal of Discrete Algorithms*, 13:67–77, 2012.

[27] M. Caserta. *Metaheuristics: Progress in complex systems optimization*, pages 43–63. Springer, Berlin, 2007.

[28] S. Ceria, P. Nobili, and A. Sassano. A Lagrangian-based heuristic for large-scale set covering problems. *Mathematical Programmimg*, 81:215–228, 1998.

[29] S. Chikkerur, A.N. Carwright, and V. Govindaraju. K-plet and coupled BFS: A graph based fingerprint representation and matching algorithm. *Advances in Biometrics, Lecture Notes in Computer Science*, 3832:309–315, 2005.

[30] V. Chvatal. A greedy-heuristic for the set covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.

[31] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18:265–298, 2004.

[32] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. In *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 149–159, 2001.

[33] A. Dharwadker. The vertex cover algorithm. *Proceedings of the Institute of Mathematics*, 2006.

[34] I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162:439–485, 2005.

[35] E. D. Dolan and J.J. More. Benchmarking optimization software with performance profiles. *Mathematical Programming Series A*, 91:201–213, 2002.

[36] F. Dorn. Planar subgraph isomorphism revisited. In *International Symposium on Theoretical Aspects of Computer Science(STACS)*, pages 263–274, Nancy, France, 2010.

[37] D. Eppstein. Subgraph isomorphism in planar graphs and related problems. In *Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 632–640, Philadelphia,USA, 1995.

[38] G. Even, D. Rawitz, and S. Shahar. Hitting sets when the VC-dimension is small. *Information Processing Letters*, 95:358–362, 2005.

[39] T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[40] M. Finger, T. Stützle, and H. Lourenço. Exploiting fitness distance correlation of set covering problems. *Lecture Notes in Computer Science*, 2279:61–71, 2002.

[41] M. L. Fisher and P. Kedia. Optimal solution of set covering/partitioning problems using dual heuristics. *Management Science*, 36(6):674–687, 1990.

[42] K. Fujisawa, T. Endo, Y. Yasui, H. Sato, N. Matsuzawa, S. Matsuoka, and H. Waki. Peta-scale general solver for semidefinite programming problems with

over two million constraints. In *The 28th IEEE International Parallel and Distributed Processing Symposium*, Phoenix, USA, 2014.

[43] B. Gallagher. Finding frequent patterns in a large sparse graph. *AAAI Fall Symposium on Capturing and Using Patterns for Evidence Detection*, pages 45–53, 2006.

[44] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[45] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[46] H. Gazit and J. H. Reif. A randomized parallel algorithm for planar graph isomorphism. *Journal of Algorithms*, 28:290–314, 1998.

[47] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfability problems using semidefinite programming. *Journal of ACM*, 42:1115–1145, 1995.

[48] F. C. Gomes, C. N. Meneses, P. M. Pardalos, and G. V. R. Viana. Experimental analysis of approximation algorithms for the vertex cover and set covering problems. *Computers and Operations Research*, 33:3520–3534, 2006.

[49] T. Grossman and A. Wool. Computational experience with aproximation algorithms for the set covering problem. *European Journal of Operational Research*, 101:81–92, 1997.

[50] N. G. Hall and R. V. Vohra. Pareto optimality and a class of set covering heuristics. *Annals of Operations Research*, 43:279–284, 1993.

[51] E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM Journal on Computing*, 31:1608–1623, 2002.

[52] Q. Han and A. P. Punnen. On the approximability of the vertex cover and related problems. *Algorithmic Aspects in Information and Management, Lecture Notes in Computer Science*, 6124:161–169, 2010.

[53] Q. Han, A. P. Punnen, and Y. Ye. An edge-reduction algorithm for the vertex cover problem. *Operations Research Letters*, 37:181–186, 2009.

[54] Q. Han and A.P. Punnen. Strong and weak edges of a graph and linkages with the vertex cover problem. *Discrete Applied Mathematics*, 160:197–203, 2012.

[55] M. Haouari and J. S. Chaouachi. A probabilistic greedy search algorithm for combinatorial optimization with application to the set covering problem. *Journal of the Operational Research Society*, 53:792–799, 2002.

[56] H. He and A. K. Singh. Graphs-at-a-time: query language and access methods for graph databases. In *SIGMOD Conference*, pages 405–418, Vancouver, Canada, 2008.

[57] C. Herrera and P. J. Zufiria. Generating scale-free networks with adjustable clustering coefficient via random walks. In *Network Science Workshop*, pages 167–172, Madrid, Spain, 2011.

[58] C. Higuera, J. C. Janodet, E. Samuel, G. Damiand, and C. Solnon. Polynomial algorithms for open plane graph and subgraph isomorphisms. *Theoretical Computer Science*, 498:76–99, 2013.

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

[60] J. Hopcroft and R. Tarjan. Efficient planarity testing. *Journal of the Association for Computing Machinery*, 21:549–568, 1974.

[61] D. K. Isenor and S. G. Zaky. Fingerprint identification using graph matching. *Pattern Recognition*, 19:113–122, 1986.

[62] L. W. Jacobs and M. J. Brusco. A local search heuristic for large set-covering problems. *Naval Research Logistics*, 42:1129–1140, 1995.

[63] J. Jaja and S. R. Kosaraju. Parallel algorithms for planar graph isomorphism and related problems. *IEEE Transactions on Circuits and Systems*, 35:304–311, 1988.

[64] H. M. Jamil. Computing subgraph isomorphic queries using structural unification and minimum graph structures. In *Symposium on Applied Computing*, pages 1053–1058, Taichung, Taiwan, 2011.

[65] J.Lee, W.S. Han, R. Kasperovics, and J.H. Lee. An in-depth comparison of subgraph isomorphism algorithms in graph databases. In *PVLDB'13 Proceedings of the 39th International Conference on Very Large Data Bases*, pages 133–144, Riva del Garda, Trento, 2013.

[66] G. Karakostas. A better approximation ratio for the vertex cover problem. In *32nd International Colloquium on Automata, Languages and Programming(ICALP)*, pages 1043–1050, Lisboa, Portugal, 2005.

[67] S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2-ϵ. *Journal of Computer and System Sciences*, 74:335–349, 2008.

[68] G. Kinney, J. W. Barnes, and B. Colleti. A group theoretic tabu search algorithm for set covering problems. Technical report, Graduate Program in Operations Research and Industrial Engineering, The University of Texas, Austin, Texas, 2004.

[69] M. Kojima, K. Fujisawa, K. Nakata, and M. Yamashita. SDPA (semidefinite programming algorithm) user's manual. Technical report, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Japan, 2005.

[70] J. P. Kukluk. Algorithm and experiments in testing planar subgraphs for isomorphism. *Journal of Graph Algorithms and Applications*, 8:101–104, 2004.

[71] G. Lan, G. W. DePuy, and G. E. Whitehouse. An effective and simple heuristic for the set covering problem. *European Journal of Operational Research*, 176:1387–1403, 2007.

[72] J. Lee, W. S. Han, R. Kasperovics, and J. H. Lee. An in-depth comparison of subgraph isomorphism algorithms in graph databases. *Proceedings of International Conference on Very Large Data Bases*, 6(2):133–144, 2012.

[73] A. Lingas. Subgraph isomorphism for biconnected outerplanar graphs in cubic time. *Theoretical Computer Science*, 63:295–302, 1989.

[74] V. Lipets, N. Vanetik, and E. Gudes. Subsea: an efficient heuristic algorithm for subgraph isomorphism. *Data Mining Knowledge Discovery*, 19:320–350, 2009.

[75] J. Llados, E. Marti, and J. J. Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:1137–1143, 2001.

[76] L. A. N. Lorena and L. S. Lopes. Genetic algorithms applied to computationally difficult set covering problems. *Journal of Operational Research Society*, 48:440–445, 1997.

[77] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 5:960–981, 1994.

[78] E. Marchiori and A. Steenbeek. An iterated heuristic algorithm for the set covering problem. In *Proceedings WAE'98*, pages 1–3, Saarbrücken, Germany, 1998.

[79] V. Melkonian. New primal-dual algorithms for steiner tree problems. *Computers and Operations Research*, 34:2147–2167, 2007.

[80] B.T. Messmer and H. Bunke. Efficient subgraph isomorphism detection: a decomposition approach. *Knowledge and Data Engineering, IEEE Transactions*, 12:307–323, 2000.

[81] N. Musliu. Local search algorithm for unicost set covering problem. *Lecture Notes in Artificial Intelligence*, 4031:302–311, 2006.

[82] İ. Muter, Ş. İ. Birbil, and G. Şahin. Combination of metaheuristic and exact algorithms for solving set covering-type optimization problems. *INFORMS Journal on Computing*, 22:603–619, 2010.

[83] H. Nagamochi and T. Ibaraki. An approximation of the minimum vertex cover in a graph. *Japan Journal of Industrial and Applied Mathematics*, 16:369–375, 1999.

[84] M. Neuhaus and H. Bunke. An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification. *Structural, Syntactic, and Statistical Pattern Recognition, Lecture Notes in Computer Science*, 3138:180–189, 2004.

[85] M. Neuhaus and H. Bunke. A graph matching based approach to fingerprint classification using directional variance. *Audio-and Video-Based Biometric Person Authentication, Lecture Notes in Computer Science*, 3546:191–200, 2005.

[86] M. Neuhaus, K. Riesen, and H. Bunke. Fast suboptimal algorithms for the computation of graph edit distance. *Structural, Syntactic, and Statistical Pattern Recognition, Lecture Notes in Computer Science*, 4109:163–172, 2006.

[87] M. Okun. On approximation of the vertex cover problem in hypergraphs. *Discrete Optimization*, 2:101–111, 2005.

[88] S. Poljak. A note on stable sets and coloring of graphs. *Commun. Math. Univ. Carolinae*, 15:307–309, 1974.

[89] Z. G. Ren, Z. R. Feng, L. J. Ke, and Z. J. Zhang. New ideas for applying ant colony optimization to the set covering problem. *Computers and Industrial Engineering*, 58:774–784, 2010.

[90] C. R. Rivero, I. Hernandez, D. Ruiz, and R. Corchuelo. Benchmarking data exchange among semantic-web ontologies. *IEEE Transactions on Knowledge and Data Engineering*, 25:1997–2009, 2013.

[91] C. R. Rivero and H. M. Jamil. Exact subgraph isomorphism using graphlets and minimum hub covers. Submitted to PVLDB for publication, 2014.

[92] C.R. Rivero and H. M. Jamil. On isomorphic matching of large disk resident graphs using an XQuery engine. International Workshop on Graph Data Management: Techniques and Applications.

[93] M.D. Santo, P. Foggia, C. Sansone, and M. Vento. A large database of graphs and its use for benchmarking graph isomorphism algorithms. *Pattern Recognition Letters*, 24:1067–1079, 2003.

[94] J. Saramaki and K. Kaski. Scale-free networks generated by random walkers. *Physica A: Statistical Mechanics and its Applications*, 341:80–86, 2004.

[95] B. Saux and H. Bunke. Feature selection for graph-based image classifiers. *Pattern Recognition and Image Analysis, Lecture Notes in Computer Science*, 3523:147–154, 2005.

[96] T. Saxena, P. Tu, and R. Hartley. Recognizing objects in cluttered images using subgraph isomorphism. In *Proceedings of the IU Workshop*, California, USA, 1998.

[97] H. Shang, Y. Zhang, X. Lin, and J. X. Yu. Taming verification hardness: An efficient algorithm for testing subgraph isomorphism. In *Journal Proceedings of the Very Large Database Endowment*, volume 1, pages 364–375, Auckland, New Zealand, 2008.

[98] Y. Tian and J. M. Patel. Tale: A tool for approximate large graph matching. *International Conference on Data Engineering*, pages 963–972, 2008.

[99] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23:31–42, 1976.

[100] S. Umetani and M. Yagiura. Relaxation heuristics for the set covering problem. *Journal of the Operations Research Society of Japan*, 50:350–375, 2007.

[101] V.N. Vapnik. *Statistical Learning Theory.* Wiley-Interscience, New York, USA, 1989.

[102] F. J. Vasko and G. R. Wilson. An efficient heuristic for large set covering problems. *Naval Research Logistics Quarterly*, 31:163–171, 1984.

[103] V. V. Vazirani. *Theoretical aspects of computer science*, pages 198–207. Springer, Verlag Lecture Notes in Computer Science 2292, 2002.

[104] M. Weber, M. Liwicki, and A. Dengel. Faster subgraph isomorphism detection by well-founded total order indexing. *Pattern Recognition Letters*, 33:2011–2019, 2012.

[105] D.P. Williamson. The primal-dual method for approximation algorithms. *Mathematical Programming*, 91:447–478, 2002.

[106] M. Yagiura, M. Kishida, and T. Ibaraki. A 3-flip neighborhood local search for the set covering problem. *European Journal of Operational Research*, 172:472–499, 2006.

[107] R. B. Yates and G. Valiente. An image similarity measure based on graph matching. In *Seventh International Symposium String Processing and Information Retrieval*, pages 28–38, Curuna, Spain, 2000.

[108] B. Yelbay. Primal-dual heuristics for solving the set covering problem. Master's thesis, Sabancı University, Istanbul,Turkey, 2010.

[109] B. Yelbay, Ş. İ. Birbil, K. Bülbül, and H. M. Jamil. Trade-offs computing minimum hub cover toward optimized graph query processing, 2013. `http://arxiv.org/abs/1311.1626`.

[110] B. Yelbay, Ş.İ. Birbil, and K. Bülbül. The set covering problem revisited: An emipirical study of the value of dual information. To appear in Journal of Industrial Management and Optimization, 2014.

[111] B. Yelbay, Ş.İ. Birbil, K. Bülbül, and H. Jamil. Approximating the minimum hub cover problem on planar graphs. Submitted for publication, 2014.

[112] S. Zhang, S. Li, and J. Yang. GADDI: distance index based subgraph matching in biological networks. In *International Conference on Extending Database Technology*, pages 192–203, Saint Petersburg, Russia, 2009.

[113] P. Zhao and J. Han. On graph query optimization in large networks. *Proceedings of International Conference on Very Large Data Bases*, 3(1):340–351, 2010.

[114] K. Zhu, Y. Zhang, X. Lin, G. Zhu, and W. Wang. A novel and efficient framework for finding subgraph isomorphism mappings in large graphs. In *15th International Conference on Database Systems for Advanced Applications*, pages 140–154, Tsukuba, Japan, 2010.