INTEGER PROGRAMMING MODELS FOR THE BRANCHWIDTH PROBLEM

A Dissertation

by

ELIF ULUSAL

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2008

Major Subject: Industrial Engineering

INTEGER PROGRAMMING MODELS FOR THE BRANCHWIDTH PROBLEM

A Dissertation

by

ELIF ULUSAL

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2008

Major Subject: Industrial Engineering

ABSTRACT

Integer Programming Models
for the Branchwidth Problem. (May 2008)

Elif Ulusal, B.S., Bilkent University

Co–Chairs of Advisory Committee: Dr. Sergiy I. Butenko
Dr. Illya V. Hicks

We consider the problem of computing the branchwidth and an optimal branch decomposition of a graph. Branch decompositions and branchwidth were introduced in 1991 by Robertson and Seymour and were used in the proof of Graph Minors Theorem (GMT), a well known conjecture (Wagner's conjecture) in graph theory. The notions of branchwidth and branch decompositions have been proved to be useful for solving many NP-hard problems that have applications in fields such as graph theory, network design, sensor networks and biology. Branch decompositions have been utilized for problems such as the traveling salesman problem by Cook and Seymour, general minor containment and the branchwidth problem by Hicks by means of the relevant branch decomposition-based algorithms.

Branch decomposition-based algorithms are fixed parameter tractable algorithms obtained by combining dynamic programming techniques with branch decompositions. The running time and space of these algorithms strongly depend on the width of the utilized branch decomposition. Thus, finding optimal or close to optimal branch decompositions is very important for the efficiency of the branch decomposition-based algorithms. Motivated by the vastness of the fields of application, we aim to increase the efficiency of the branch decomposition-based algorithms by investigating effective techniques to find optimal branch decompositions.

We present three integer programming models for the branchwidth problem. Two similar formulations are based on the relationship of branchwidth problem with a special case of the Steiner tree packing problem. The third formulation is based on the notion of laminar separations. We utilize upper and lower bounds obtained by heuristic algorithms, reduction techniques and cutting planes to increase the efficiency of our models. We use all three models for the branchwidth problem on hypergraphs as well. We compare the performance of three models both on graphs and hypergraphs.

Furthermore we use the third model for rank-width problem and also offer a heuristic for finding good rank decompositions. We provide computational results for this problem, which can be a basis of comparison for future formulations.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

In this dissertation, we study effective techniques for finding the branchwidth and an optimal branch decomposition of a graph. We focus on integer programming models of this problem, which is referred to as the branchwidth problem. Integer programming is a tool for solving combinatorial optimization problems to optimality. Many combinatorial optimization problems are NP-hard, which is evidence to the potential intractability of those problems. But, in some cases, there is an absolute need for exact solutions, and in some cases, exact solutions are strongly desirable. Thus, designing fast exponential or sub-exponential algorithms, if possible, has been an interesting and growing research area.

Since the early 1990's, many width parameters and their related graph decompositions have been studied to develop fast sub-exponential algorithms for solving NP-hard problems modeled on graphs. Some of these parameters were introduced in the proofs of important graph theoretic results. The treewidth, branchwidth, pathwidth, carvingwidth, clique-width, rank-width, etc. are some examples. Researchers studied the advantages of each parameter and their best use. These width parameters have been proved to be of great use both in theory and practice.

Branch and tree decompositions, and their connectivity invariants branchwidth and treewidth were introduced by Robertson and Seymour in 1991 [79]. It has been shown that treewidth is beneficial for solving NP-complete problems that are modeled on graphs with bounded treewidth, using dynamic programming techniques [1, 19]. The benefits of treewidth also apply to branchwidth since branchwidth and

---

The journal model is *Mathematical Programming*.

treewidth of a graph bound each other by constants [79]. Thus, one can develop a sub-exponential algorithm, if possible, for some NP-complete problem using branchwidth and dynamic programming. Although treewidth and branchwidth have similar functions in related sub-exponential algorithms, it is easier to accomplish many proofs in Graph Minors Theorem (GMT) when branchwidth is used instead of treewidth. Also, branchwidth can be naturally generalized to hypergraphs and matroids. Geelen et al. [39] generalized Robertson and Seymour's [78, 79] theory on graphs to matroids by using branchwidth. Recently, Hliněný et al. [57] explained the advantages of branchwidth over treewidth.

Branchwidth and branch decompositions can be useful for solving many NP-hard problems that have applications in fields such as graph theory, network design, sensor networks, and biology. Many sub-exponential algorithms have been developed to solve different NP-hard problems using branch decompositions [16, 17, 49, 50, 34]. These algorithms are called branch decomposition-based algorithms. Some applications have been shown to be practical. However, the drawback is that for some input integer k, testing if a general graph has branchwidth at most k was proved to be NP-complete [81]. Unfortunately, the running time and space of the branch decomposition-based algorithms strongly depend on the width of the branch decomposition utilized in the algorithm. It is highly desirable to construct good branch decompositions, those of small width. Hence, it is necessary to find exact or close to optimal solutions for the branchwidth problem.

I.1. Objectives

Motivated by the vastness of the fields of application, we investigate efficient techniques to find optimal branch decompositions. The overall objective is to develop and implement integer programming models to construct optimal branch decompositions. This research comprises a series of linked objectives. These are:

1. Develop and implement three models for branchwidth problem:
    - An integer programming model that utilizes the relationship between the branchwidth problem and the Steiner tree packing problem (IP1),
    - Extension of IP1 with fewer variables and constraints (IP2), and
    - An integer programming model that benefits from laminar separations (IP3).
2. Explore the polyhedral structure and introduce local and global cuts.
3. Implement upper and lower-bounding techniques in conjunction with the integer programming models.
4. Implement reduction techniques on the input graph for the branchwidth problem.
5. Compare the overall effectiveness of the three models.
6. Implement an extension of IP3 for rank-width problem.
7. Implement a heuristic to obtain a rank decomposition and an upper bound for the rank-width.
8. Implement reduction techniques on the input graph for the rank-width problem.

I.2. Contribution

We offer three integer programming formulations, two of which are similar, that compute the branchwidth and construct an optimal branch decomposition of a general graph. First two models utilize a Steiner tree packing formulation on a new graph constructed using the given input graph. The last model benefits from the laminar separations (defined later). All three models can be used to compute the branchwidth and an optimal branch decomposition of a hypergraph as well. The third model can also be used for the computation of the branchwidth and construction of an optimal branch decomposition of symmetric submodular set functions. For instance, rank-width of a graph, a graph complexity measure introduced by Oum and Seymour [72], can be computed and its related decomposition can be constructed using the last model. The rank-width is defined to be the branchwidth of the cut-rank function, a symmetric submodular function, of a graph. We implement all three models for general graphs and hypergraphs. Also, we implement the third model for the rank-width problem.

In addition, we explore the polyhedral structure of the first two models to introduce cuts. We implement some heuristics to obtain upper and lower bounds for all three models. These heuristics help reduce the run times and increase efficiency considerably. We use some reduction techniques and some anti-symmetry constraints to reduce the run times even more. Moreover, we prove some important results on laminar separations that leads us to the construction of the third model. Also, for the implementation of the third model on the rank-width problem, we apply some appropriate reductions on the input graph, and we implement a rank-width heuristic to obtain upper bounds for the rank-width of the test instances.

The proposed integer programming models provide an optimal branch decomposition and the branchwidth of a given graph, in a reasonable amount of time, for small graphs or hypergraphs. These models can be useful in decreasing the running time of the branch decomposition-based algorithms for combinatorial optimization problems by means of the obtained optimal branch decomposition. To the best of our knowledge, our models are the first integer programming models for the branchwidth problem. The computational results we obtained demonstrate the difficulty of solving this problem to optimality and will be a basis of comparison for the future models of the branchwidth problem. In addition, we have some computational results for the rank-width problem, which will be a basis of comparison for the future models as well. To the best of our knowledge, the application of the third model on rank-width problem is also the first integer programming formulation for this problem.

I.3. Organization

The first part of Chapter II reviews the basic graph definitions. The other sections give the definitions of branch decompositions, Steiner tree packing, and rank decompositions. Chapter III offers a literature review of the research done on branchwidth and branch decompositions, including their theoretical and practical benefits in the fields of graph theory and operations research. Also, it gives a short literature review on rank-width, rank decompositions, and some different versions of the Steiner tree packing problem.

Chapter IV starts with presenting the necessary notations, and then explains the relationship between the branchwidth problem and a special case of the Steiner tree packing problem. Also, it gives a formulation of the multicast congestion problem, which is the closest problem to the special case we deal with in our study. Then, it presents the first two formulations. Furthermore, it describes the upper and lower bound heuristics that we have applied to our models, the preprocessing procedures, and the cuts we have implemented for the first two models. Moreover it covers the implementation of the first two models on hypergraphs. Finally, we offer computational results we have obtained using the first two models and some comments on the comparison of the two models both on graphs and hypergraphs.

Chapter V describes the third formulation and covers the upper and lower bound heuristics we have implemented for the third model, as well as the implementation of the third model on hypergraphs. The chapter also offers computational results for the third model and their comparison with the results of the previous two models. Chapter VI reviews vertex-minors, a related notion to the rank-width that was introduced by Oum [69], and also explains how we can implement the third model for the rank-width problem. Furthermore, we explain the preprocessing steps and a rank-width heuristic that we have implemented and offer computational results for the rank-width problem. Finally, Chapter VII is reserved for conclusions and ideas for future work.

CHAPTER II

PRELIMINARIES

II.1. Basic definitions

A *graph* is an ordered pair (V, E) where V is a nonempty set, referred to as the set of *vertices* or *nodes*; E, the set of *edges*, is an unordered binary relation on V. A *hypergraph* is an ordered pair (V, E) of nodes and edges, and an incidence relationship between them that is not restricted to two ends for each edge. A graph is a hypergraph where all edges have at most two ends. A *directed graph* is a graph where the edge set is composed of ordered pair of vertices, which we refer to as *directed edges*. An *undirected graph* is a graph with no directed edges. A *loop* is an edge (directed or undirected), which starts and ends on the same vertex. *Multiple edges* or *parallel edges* in a graph are two or more edges that are incident to the same two vertices. A *simple graph* is a graph with no loops and no multiple edges. All of our graphs are simple and undirected, unless stated otherwise.

A *walk* in a graph is a sequence of vertices such that from each of its vertices there is an edge to the next vertex in the sequence. A *path* is a walk with no repeated vertices. A *cycle* is a closed walk with no other repeated vertices than the starting and ending vertices. A graph is *connected*, if every pair of vertices can be joined by a path. A graph that does not contain any cycle is called a *forest*. A connected forest is called a *tree*. The *degree* of a vertex v is the number of edges incident with v and is denoted by *deg(v)*. The *leaves* of a tree are the vertices of degree one. We refer to the nodes that are not leaves as *non-leaf nodes*. A graph is called *complete*, if there is an edge between every pair of vertices of the graph. A *biconnected* graph is a connected graph that is not divided into disconnected pieces when any single vertex (and incident edges) is deleted. A graph is *planar*, if it can be embedded in a plane such that no two edges cross.

The complement of a graph $G = (V, E)$ is a graph $\bar{G} = (V, \bar{E})$ such that two vertices in G are adjacent if and only if they are not adjacent in G. A graph $\hat{G} = (\hat{V}, \hat{E})$ is a *subgraph* of the graph $G = (V, E)$ if $\hat{V} \subseteq V$ and $\hat{E} \subseteq E$. For a subset $V' \subseteq V$, $G[V']$ denotes the subgraph induced by V', i.e., $G[V'] = (V', E \cap E')$, where E' represents the edges of the complete graph on V'. *Contraction* of an edge e means deleting that edge and identifying the ends of e into one node. The parallel edges are identified; and loops are deleted in the contraction process. A graph H is a *minor* of a graph G if H can be obtained from a subgraph of G by a series of operations, such as deletion of vertices, deletion of edges or contractions.

The *connectivity* of a graph is the smallest number of vertices which can be removed to disconnect the graph. Similarly, the *edge connectivity* of a graph is the smallest number of edges which can be removed to disconnect the graph. A graph is called *2-edge connected* if its edge connectivity is 2 or more.

II.2. Branch decompositions

Let $G = (V, E)$ be a hypergraph and T be a ternary tree (a tree where every non-leaf node has degree 3) with $|E(G)|$ leaves. Let ν be a bijection from the edges of G to the leaves of T. Then the pair (T, ν) is defined to be a *branch decomposition* of G by Robertson and Seymour [79].

A *separation* of a graph G is a pair (G_1, G_2) of subgraphs with $G_1 \cup G_2 = G$ and $E(G_1 \cap G_2) = \emptyset$, and the *order* of this separation is defined as $|V(G_1 \cap G_2)|$. An example graph G with 6 vertices and 9 edges, and a separation of order 3 for G are shown in Figure 1. The rest of the figures in this section will be based on the example graph of Figure 1.

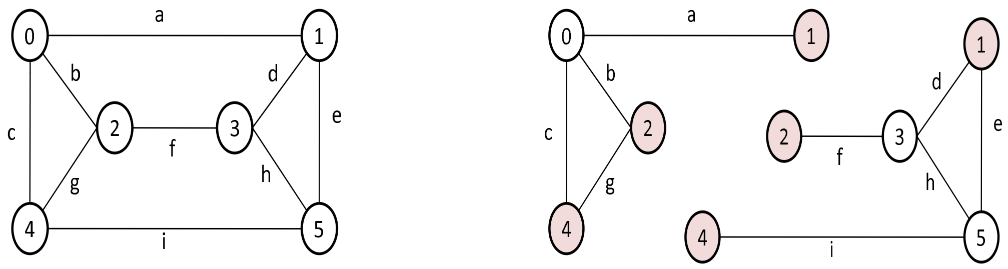


Fig. 1 : G (left) and a separation of order 3 (right)

Let (T, ν) be a branch decomposition. Then, removing an edge, say q, from T partitions the edges of G into two subsets A_q and B_q. Figure 2 shows a branch decomposition tree T of G on the left, and the separation generated by removing the edge p from T on the right.

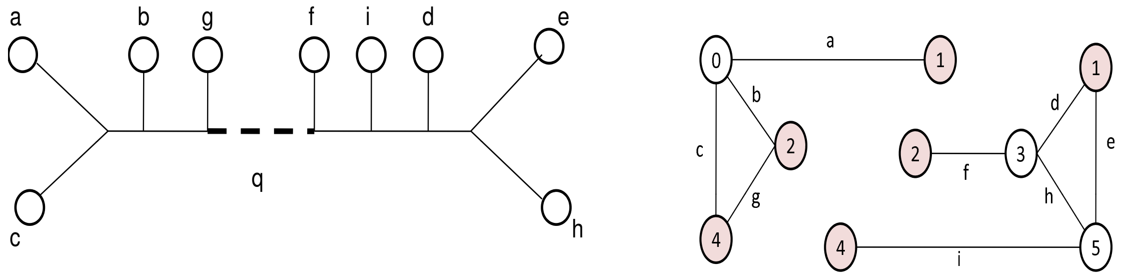


Fig. 2 : A branch decomposition tree T (left) of G and a separation of G generated by removing edge q from T (right)

The *width* of an edge q is the number of vertices of G that are incident with at least one edge in both sets A_q and B_q. These intersection vertices are called the *middle set* of q and denoted by *mid(q)*. The width of edge q is equal to the order of the separation $(G[A_q], G[B_q])$. The width of edge q in Figure 2 is 3, and the middle set $mid(q) = \{1, 2, 4\}$. The *width* of a branch decomposition (T, ν) is the maximum width among all edges of the branch decomposition. The *branchwidth* of G, denoted by $\beta(G)$, is the minimum width over all branch decompositions of G. A branch decomposition of G with width equal to the branchwidth is an *optimal branch decomposition* of G. Figure 3 shows two branch decompositions of G. The branchwidth of G is 3. With a few calculations, we can easily see that the maximum width among all the edges of these branch decompositions is 3. Thus, they are both optimal branch decompositions.

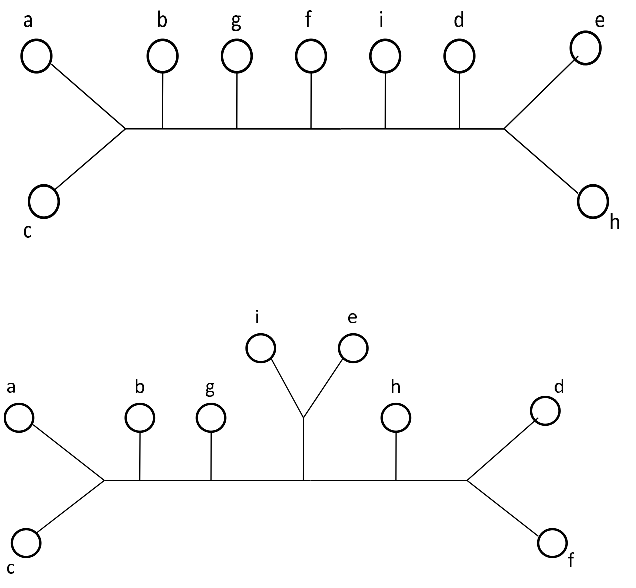


Fig. 3 : Two different optimal branch decompositions of G

A branch decomposition (T, ν) is called *connected*, if for every edge $e \in E(T)$, the two edge-induced subgraphs of G corresponding to the two components of $T \backslash e$ are both connected. The branch decompositions in Figure 3 are both connected. A *partial branch decomposition* is a branch decomposition without the restriction of every non-leaf node having degree 3. The branchwidth problem is to find the branchwidth and an optimal branch decomposition.

II.3. Steiner tree packing

Let $H = (V_H, E_H)$ be an undirected graph with positive integer capacities c_e and nonnegative lengths w_e for each edge e in E_H. Let $\Phi = \{N_i : i = 1, ..., k\}$ be a family of subsets of V_H. Each subset N_i defines a *net*; and Φ is called a *netlist*. The nodes in a net are called *terminals* of that net.

A graph *spans* a vertex set, if that graph contains every vertex in that set. Let T_i be a tree that spans N_i. Then, the set $\Gamma = \{T_i : i = 1, ..., k\}$ is a family of trees, which is called a *routing*. The tree T_i is called a *Steiner tree* for N_i. We define the *load* or *congestion* $L(\Gamma, e)$ on an edge e as the number of trees in Γ that contain edge e. If the load of an edge is larger than the capacity of that edge, then the edge is *over-saturated*. A routing that has no over-saturated edge is called a *legal routing*. The *Steiner tree packing problem* is to find a legal routing[1]. Figure 4 shows a routing for a netlist consisting of two nets. Assuming the edge capacities in the input graph to be 1, the routing in the figure is not a legal routing, since there is an edge of load two. If the edge capacities are 2, then the same routing is a legal routing.

The problem of finding a routing in an input graph for a netlist is called the *global routing problem*. There are different versions of the global routing problem, which also contains the variations of the Steiner tree packing problem. Routing problems can be constrained or unconstrained. Finding a legal routing of minimum length, i.e., sum of the lengths of all edges in the routing, is called the *constrained global routing problem*. In this version of the problem, over-saturation of the edges is not allowed. A second version of the problem is finding the minimum length routing among those that minimize the maximum over-saturation on an edge, which is referred to as an

[1]Most of the above terminology is taken from Chopra [15] and Lengauer and Lügering [63].

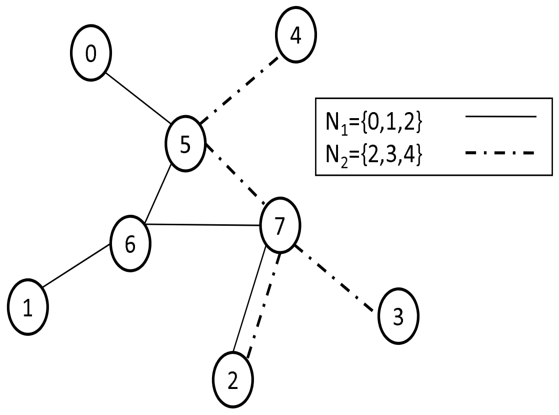


Fig. 4 : A routing of two nets

unconstrained global routing problem. Both versions of the problem are NP-hard since they include a special case of the problem of finding a minimum length Steiner tree, which is known to be NP-hard [35].

In our research, we focus on a slightly different version of the problem, where the aim is to minimize the maximum load on an edge only, not the total length of the routing. It is a special case of unconstrained global routing problem, where the edge capacities are 1 and the lengths are 0. It is referred to as the multicast congestion problem in the literature. We call the minimum of maximum loads of all routings as the *minmax load.* Let $\kappa(H, \Phi)$ be the minmax load on graph H for netlist Φ. Assuming that the input graph H is K_5, the complete graph on 5 vertices, Figure 5 shows an example of an optimal routing for a multicast congestion problem on H, with a netlist Φ of 3 nets. Each type of line represents a Steiner tree that spans a different net. Maximum congestion is 1 in this routing, and this is an optimal routing. Thus, $\kappa(H, \Phi) = 1$.

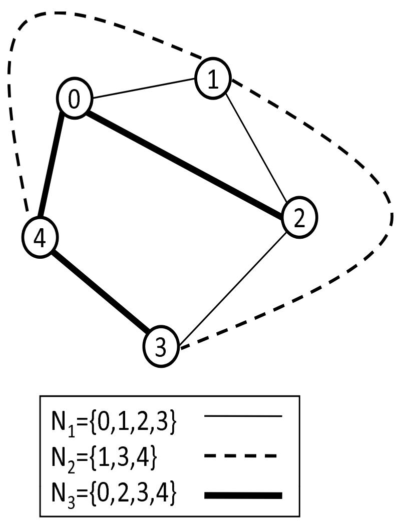
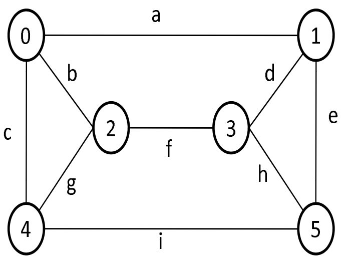


Fig. 5 : An optimal routing for a multicast congestion problem

II.4. Rank decompositions

Let $\mathbb{Z}$ be the set of integers. For a finite set S, a function $f : 2^S \to \mathbb{Z}$ is called *symmetric* if $f(X) = f(S \setminus X)$ for all $X \subseteq S$. If $f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y)$ for all subsets X, Y of S then f is called *submodular*. Let T be a ternary tree and μ be a bijection from the elements of S to the leaves of T. Then the pair (T, μ) is called a *branch decomposition* of the symmetric submodular set function f. Then removing an edge, say e, from T partitions the elements of S into two subsets X and Y. The *width* of edge e is $f(X)$. The *width* of a branch decomposition (T, μ) is the maximum width among all edges of the branch decomposition. The *branchwidth* of f, denoted by $\beta(f)$, is the minimum width over all branch decompositions of f.

We use the notation used by Hliněný and Oum [56] to define rank-width. For an $|X| \times |Y|$ matrix R (where rows are indexed by X and the columns are indexed by Y) and $A \subseteq X$, $B \subseteq Y$, let $R[A, B]$ be the $|A| \times |B|$ submatrix of R. For a graph G, let $Adj(G)$ be the adjacency matrix of G, that is a $|V| \times |V|$ binary matrix. An entry is 1 in $Adj(G)$ if and only if there exists an edge in G between the vertices corresponding to the column and the row of that entry. The *cut-rank* function $\rho_G(X)$ of a graph $G = (V, E)$ is defined as the rank of the matrix $Adj(G)[X, V \setminus X]$ for each subset X of V. $\rho_G(X)$ is symmetric and submodular [72]. The optimal branch decomposition and branchwidth of the cut-rank function of G are called the *optimal rank decomposition* and the *rank-width* of G. We illustrate an optimal rank decomposition of the example graph $G = (V, E)$ of Figure 1. Let $X = \{0, 1\} \subseteq V$. Figure 6 shows G, cut rank function of $X \subseteq V$ and an optimal rank decomposition of G.



$$rank\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} = 2$$

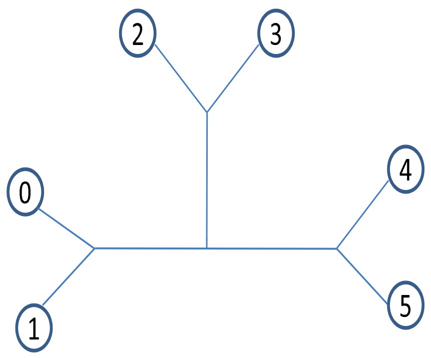


Fig. 6 : G (top), the cut rank function for the vertex set $X = \{0, 1\}$ (left) and the optimal rank decomposition of G (right)

CHAPTER III

LITERATURE REVIEW

III.1. Branchwidth and branch decompositions

Branch decomposition-based algorithms have been proposed for ring routing [16], traveling salesman [17], general minor containment [49], optimal branch decomposition [50], independent set [34], dominating set [34], longest cycle [34], bisection [34], and (k, r)-center problems [25]. For instance, Demaine et al. [25] showed that there is a polynomial time approximation scheme for the r-domination problem in both planar graphs and map graphs, using branch decomposition-based algorithms.

As a practical example, Cook and Seymour [17] utilized branch decompositions to obtain high-quality tours for the traveling salesman problem by merging collections of tours produced by some traveling salesman heuristics. They show that branch decomposition-based algorithms can be practical in large-scale optimization. They offered computational results for graphs with more than 10,000 nodes. In addition, Hicks [49] implemented a branch decomposition-based algorithm for the minor containment problem based on the framework given by Seymour and Thomas [80]. Given a simple graph G and a simple connected graph H this algorithm checks if H is a minor of G. He utilized some heuristics to compute the branch decomposition that is used in this algorithm [49]. Interestingly, a pertinent practical aspect of this algorithm is that, it can be used to find lower bounds for the branchwidth of non-planar graphs. Moreover, Hicks [50] implemented a practical branch decomposition-based algorithm for finding optimal branch decompositions.

Branch decomposition-based algorithms for combinatorial optimization problems modeled on graphs of bounded branchwidth can be summarized as follows. First, find a branch decomposition of small width, and then use a polynomial-time algorithm that solves the instances of bounded branchwidth. This procedure is promising for finding fast algorithms. But, there are two important points about this procedure. One is that not all optimization problems can be solved in polynomial time using this procedure, e.g. the bandwidth minimization, which is proved to be NP-hard even on ternary trees [36, 68]. The other point is that the running time and space of these algorithms strongly depend on the width of the branch decomposition constructed in the first step. The second step usually runs in exponential time in the width of that branch decomposition. Unfortunately, testing if a general graph has branchwidth at most k, some input integer, has been proved to be NP-complete [81].

Seymour and Thomas [81] proved that the branchwidth of a planar graph can be computed in polynomial time and proposed an algorithm for computing the branchwidth of planar graphs. Hicks [51] implemented this algorithm. It computes only the branchwidth of a planar graph. Hicks also implemented [52] another algorithm based on the framework of Seymour and Thomas [81], improving the first algorithm with the computation of an optimal branch decomposition. Also a linear time heuristic was proposed for planar graphs by Tamaki [82]. Moreover, Gu and Tamaki [45] provided an $O(n^3)$ time algorithm for planar graphs, where n is the number of vertices. Later, Bian et al. [5] offered efficient implementations of the procedure presented by Seymour and Thomas [81]. Their implementation finds the branchwidth of planar graphs up to 100,000 edges in a practical time and memory space. Also, for planar graphs Fomin and Thilikos [34] proved that the branchwidth is at most $2.122\sqrt{n}$, where n is the number of vertices.

For general graphs, Robertson and Seymour offered the first heuristic which is fast, but estimates branchwidth within a factor of 3. For instance, it decides either that a graph has branchwidth at least 10 or finds a branch decomposition of width at most 30. This approximation algorithm is not practical. Although researchers improved this method, the resulting algorithms were impractical as well [62, 65, 77, 6, 7].

In 1994, Cook and Seymour [16] offered a heuristic called the eigenvector method for general graphs, which is based on an eigenvector technique for finding graph separators. Later in 2003, they showed that it is practical by using it in a branch decomposition-based algorithm for the traveling salesman problem(TSP) [17]. Two other heuristics have been offered to find good branch decompositions, (i.e., branch decompositions of small width) by Hicks [48]. One of the methods is called diameter method and the other, referred to as the hybrid method, is a combination of the eigenvector and diameter methods [48]. These three methods do not have a worst case bound but perform well in general experimentally. Demaine et al. [24] proposed a polynomial time algorithm to approximate branchwidth within a factor of 2.25. Fomin and Thilikos [33] provided an upper bound for branchwidth based on its minimum dominating set and genus.

Bodlaender and Thilikos [9] offered a linear time fixed parameter tractable algorithm that decides if a graph has branchwidth at most k, for any constant k, and if so constructs the corresponding branch decomposition, given a similar structure to the branch decomposition as an input. Although this method is not practical, it leads to a practical algorithm for $k \leq 3$. In 1999, Bodlaender and Thilikos [10] offered an algorithm that decides if the branchwidth is less than 3, and if so, computes an optimal branch decomposition.

In addition, Kloks et al. [60] presented an algorithm that computes the branchwidth of interval graphs with n vertices in $O(n^3 logn)$ time, and they showed that this method can be generalized for permutation and trapezoid graphs. For split graphs and bipartite graphs they showed that it is NP-Complete to compute the branchwidth [60]. Also, Mazoit [66] proved that the branchwidth of circular-arc graphs can be computed in polynomial time. Fomin et al. [31] proposed an algorithm that computes the branchwidth of a general graph in $O(2+\sqrt{3})^n \cdot n^{O(1)}$ time, where n is the number of vertices. Paul and Telle [76] introduced new structures that lead to fast and simple algorithms to compute branchwidth. In addition, Hicks [50] implemented the first practical algorithm that computes the branchwidth and an optimal branch decomposition for general graphs. Given a graph G and a branch decomposition of G of width $k \geq 3$, his branch decomposition-based algorithm, by repeatedly lowering k, decides if the graph has branchwidth at most $k-1$, until it finds an optimal branch decomposition.

Furthermore, there have been some studies on characterization of graphs of bounded branchwidth. A class of graphs $\mathcal{C}$ is called *minor closed* if all the minors of any member of $\mathcal{C}$ also belong to $\mathcal{C}$. Given a minor closed class of graphs $\mathcal{C}$, the *obstruction set* of $\mathcal{C}$ is the set $\mathcal{Q}$ of graphs such that none of the members of $\mathcal{Q}$ belongs to $\mathcal{C}$, but all of their proper minors do. Robertson and Seymour [79] proved that for a given integer k, the class of graphs with branchwidth at most k is a minor closed class. Thus, if G has branchwidth at most k, then any minor of G has branchwidth at most k as well. This implies that the class of graphs with branchwidth at most k has a finite obstruction set. The only member of the obstruction set for graphs with branchwidth at most 2 is K_4 (4-clique) [79]. The obstruction set for graphs with branchwidth at most 3 consists of K_5 (5-clique), Q_3 (cube3), M_6 (octahedron), and M_8 (Möbius ladder) [10].

In addition, Robertson and Seymour [79] showed that $n \times n$-grid graphs have branchwidth n, and cliques have branchwidth $\lceil (2/3)|V(G)| \rceil$. The branchwidth of chordal graphs satisfies $\lceil (2/3)\omega(G) \rceil \leq \beta(G) \leq \omega(G)$ where $\omega(G)$ is the maximum clique number of graph G [79, 47], and $\beta(G)$ is the branchwidth of G. A graph has branchwidth at most three if and only if it does not contain the aforementioned members of its obstruction set as a minor [10]. Also, Paul et al. [75] studied the edge-maximal graphs of branchwidth k.

Moreover, in his PhD thesis, Dorn [26] introduced the special branch decompositions, which link branch decompositions and tree decompositions. He also introduced 3-width, which can be defined using branch decompositions, and proved that the treewidth of a graph is equal to its 3-width and showed that any branch decomposition-based algorithm can be transferred to a tree decomposition-based algorithm that has the same running time, and vice versa [26]. Later, Dorn et al. [27] used dynamic programming on special branch decompositions to design fast subexponential algorithms for several graph problems.

In addition, Dorn and Telle [29] introduced semi-nice tree decompositions, which are smaller structures similar to tree decompositions that combine the best of branchwidth and treewidth. They introduce simple algorithms to transform branch and tree decompositions to semi-nice tree decompositions, and showed that for some optimization problems semi-nice tree decomposition-based algorithms give better running times than branch and tree decomposition-based algorithms [29]. Dorn [28] also proposed a new approach for solving NP-hard problems by combining dynamic programming on branch decompositions with matrix multiplication and showed that this approach works faster than the dynamic programming algorithms for any vertex subset problem on graphs of bounded branchwidth.

Branchwidth notion can also be extended to matroids. It has been shown that branchwidth and branch decompositions are useful in producing matroid analogs of Graph Minors Theorem (GMT) [39]. Hicks and McMurray [53], and idependently Mazoit and Thomassě [67], showed that the branchwidth of a bridgeless graph is equal to the branchwidth of its cycle matroid. Robertson and Seymour [79] characterized the class of matroids with branchwidth at most 2 and Geelen et al. [37] proved that the size of excluded minors for the class of matroids with bounded branchwidth is finite. Hall et al. [46] showed that the number of excluded minors for matroids of branchwidth 3 is no more than 14. Then, Hliněný [54] proved that this number is exactly 10. Also, Geelen et al. [38] studied the obstructions to branch decomposition of matroids and obtained a qualitative characterization of matroids of large branchwidth.

Moreover, in 2005, Hliněný [55] presented an algorithm that, for a given matroid M of bounded branch-width t, finds a branch decomposition of width at most $3t$. In addition, Oum and Seymour [72] proposed a polynomial time algorithm to approximate the branchwidth of a matroid. This is an $O(n^{3.5})$ algorithm, which, given an integer constant k, decides if a matroid M has branch-width at most k, and if so finds a branch-decomposition of width at most $3k - 1$. Later, Oum and Seymour [73] presented a more precise but impractical algorithm which decides if a matroid M has branchwidth k, an integer constant, and if so finds an optimal branch decomposition. Hliněný and Oum [56] presented another algorithm that runs in $O(n^3)$ time for a fixed k and gives a branch decomposition of width at most k, if there is one. Recently, Cunningham and Geelen [23] showed that an integer programming model can be solved in pseudo-polynomial time when the constraint matrix of the model is non-negative and the column matroid of this matrix has constant branchwidth.

III.2. Rank-width and rank decompositions

Rank-width is another width parameter closely related to branchwidth. Other than graphs, branchwidth can also be defined for symmetric submodular set functions.

Rank-width is defined to be the branchwidth of the cut-rank function, a symmetric submodular set function, of a graph. It is introduced by Oum and Seymour [72] in 2006 as a tool for devising an $f(k)$-expression for a graph with clique-width k, where f is a fixed function.

Clique-width is another graph invariant that was introduced by Courcelle and Olariu [20]. Like in the case of branchwidth and treewidth, many NP-hard graph problems can be solved in polynomial time, given an input graph of bounded clique-width and a corresponding decomposition, which is referred to as a k-expression. Not surprisingly, it is shown to be NP-hard to find the clique-width of a general graph [30]. On the other hand, rank-width and rank decompositions are successfully used to construct $f(k)$-expressions for the graphs of bounded clique-width. But, finding rank-width of a graph is proved to be NP-hard as well [56]. Thus, finding good rank decompositions is necessary for the success of the algorithms that use clique-width as the graph measure.

Oum [69] characterized the graphs of rank-width at most 1 and later proved that the rank-width of a graph is less than or equal to its branch-width [71]. Oum and Seymour [72] offered an algorithm that constructs a rank-decomposition of width at most $3k + 1$ for a graph of rank-width at most k. Then, Oum [70] improved this algorithm. Later, Courcelle and Oum [21] proposed a polynomial time algorithm that decides if a graph has rank-width at most k benefitting from Oum's algorithm and results [70, 69] on vertex-minors' relation with rank-width. To fill in the missing part of this algorithm, which is finding a rank decomposition of width at most k for graphs of rank-width at most k, Oum and Seymour [73] presented a polynomial time algorithm. However, this algorithm is not practical. Recently, Hliněný and Oum [56] proposed another algorithm that runs in $O(n^3)$ time for a fixed k and constructs a rank decomposition of width at most k, if there is one.

III.3. Steiner tree packing

Though not obviously related to the width parameters, the Steiner tree packing problem (STPP) is another NP-Complete problem that has received considerable attention. A more general version is stated as the global routing problem, and widely used in Very Large Scale Integrated (VLSI) design. There are different versions of the global routing problem, which also contains variations of STPP. Many versions of the problem are NP-hard since they include as a special case, the problem of finding a minimum length Steiner tree, which is known to be NP-hard [35].

Several researchers have proposed different approaches to solve the global routing problem. Lengauer and Lügering [63] and Chopra [15] studied several integer programming formulations of the global routing problem. Chopra [15] also ranked them according to their LP-relaxation lower bounds. Lengauer and Lügering [64] introduced some preprocessing techniques for this problem. Also, Terlaky et al. [83] investigated generalized models for the global routing problem and presented an approximation algorithm.

In addition, Behjat et al. [3] utilized a widely used integer programming formulation of the global routing model and proposed a matrix reordering technique that would reduce the memory requirements and the running times for solving this model. Later, Behjat et al. [4] proposed a global routing heuristic that combines several objectives for VLSI design. Grötschel et al. [40, 41, 42, 43, 44] investigated the polyhedral structure of the STPP, cutting planes, facets and separation algorithms in their series of papers and also provided computational results of a cutting plane algorithm. In addition, Koch and Martin [61] implemented a branch and bound algorithm for the same problem.

Moreover, Wang et al. [85] presented two heuristics for a different version of the

STPP. Jain et al. [58] proposed an approximation algorithm for the edge-disjoint version of the STPP. Cheriyan and Salavatipour [13, 14] studied the hardness of approximation algorithms for several versions of the STPP on directed and undirected graphs and presented a randomized algorithm for the element disjoint STPP. In 2007, Lau presented a polynomial time constant factor approximation algorithm for the edge-disjoint STPP.

The minimum multicast congestion problem is a generalization of the global routing problem. This problem has a subtle relationship with branchwidth problem, which we will explain in Chapter IV. Some approximation algorithms were offered by Vempala and Vöcking [84], Carr and Vempala [12], Baltz and Srivastav [2], Jansen and Zhang [59] for this problem. These are the only studies on minimum multicast congestion problem of which we are aware.

CHAPTER IV

MODELS 1 and 2

IV.1. Notations

Let $G = (V, E)$ be a biconnected graph with $|E| \geq 3$. For a node $v \in V$, let e_v denote a designated edge of G that is incident with v. Also, let I_v denote the set of all edges incident with v. In a branch decomposition of G, there are $|E|$ leaves since there is a bijection between the leaves of the branch decomposition tree and the edges of G. Also, there are $|E| - 2$ nodes of degree 3.

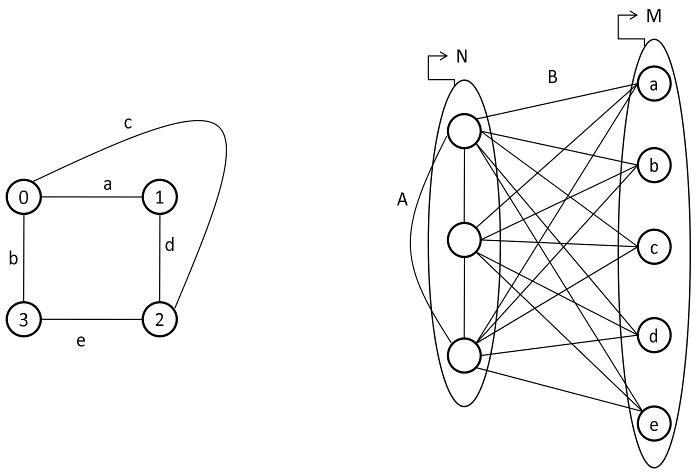


Fig. 7 : G and G'

Given G, we construct a graph G' as follows. Let N denote a set of $|E|-2$ nodes, which we refer to as the Steiner nodes, and let M be a set of $|E|$ nodes corresponding to $|E|$ edges of G by a bijection ν. Since ν is a bijection, iterators for E and M will be used interchangeably. Let A denote the set of all possible edges between each pair of vertices in N and let B denote the set of all possible edges between a member of

M and a member of N. Let $Q_v \subseteq M$ denote the set of nodes corresponding to the set of edges $I_v \subseteq E$. $G' = (V', E')$ is the graph with $V' = M \cup N$ and $E' = A \cup B$. For better illustration we use a small graph. Figure 7 shows an example graph G and the corresponding graph G' that we construct out of G.

IV.2. The relationship between branch decompositions and Steiner tree packing

In this section, we explain how the branch decomposition problem can be viewed as a Steiner tree packing problem.

Our aim is to find an optimal branch decomposition of the input graph G. A branch decomposition is a tree where each node has degree one or three. The leaf nodes are mapped to the edges in E by a bijection. Now, let G' be an input graph for a Steiner tree packing problem. The edges of G' have capacity 1 and length 0. Let each node set Q_v be a net. Let T_v be a Steiner tree that spans Q_v and $\Gamma = \{T_v \mid v \in V\}$ be a special type of Steiner tree packing in G', such that the packing forms a ternary tree.

The structure we obtain by packing $|V|$ Steiner trees on a ternary tree in G' corresponds to a branch decomposition of G. It is obvious that vertex v belongs to the middle set of some edge f if and only if f is used in a path between some pair of nodes in Q_v, i.e., f is used to construct the Steiner tree T_v. So, v belongs to the middle set of the edges in T_v. As a summary, G' being the input graph for the Steiner tree packing problem and each set of nodes corresponding to the incident edges of a vertex in G being a net, i.e., $\Phi = \{Q_v, v \in V\}$ is the netlist, a Steiner tree packing on a ternary tree in G' (where the leaves are the members of M) is a branch decomposition of G. The order of the separation induced by edge e of the branch decomposition is equal to the number of trees in the packing that employ this edge, which is defined as the load of e.

Finding an optimal branch decomposition of G is equivalent to finding a Steiner tree packing on a ternary tree T in G' that minimizes the maximum load on an edge. Note that, the version of Steiner tree packing problem that is concerned with the minimization of maximum load is the multicast congestion problem. Since we have a special Steiner tree packing, which is restricted to be on a ternary tree, this constitutes a special case of the multicast congestion problem as well. We state this relationship in the following theorem.

Theorem 1. *An optimal branch decomposition of G is equivalent to a special case of multicast congestion problem (a version of Steiner tree packing problem) on a ternary tree, a subgraph of G', for a netlist $\Phi = \{Q_v, v \in V\}$. Thus $\beta(G) = \kappa(\hat{H}, \Phi)$, $\hat{H}$ being a ternary tree and a subgraph of G'.*

IV.3. A formulation for multicast congestion

Before we start explaining our formulations, we present an integer programming model proposed for the multicast congestion problem by Baltz and Srivastav [2].

Let $G = (V, E)$ be an input graph and $\{S_1, S_2 ... S_k\}$ be a set of multicast requests. A solution is a set of k Steiner trees $\{T_1, T_2 ..., T_k\}$, where each tree T_i spans the terminals in the multicast request S_i. The congestion of an edge e of the input graph is the number of Steiner trees that contain edge e. An optimal solution is a set of k Steiner trees, each spanning a different multicast request, such that the maximum edge congestion is minimized.

Let τ_i be the set of all possible Steiner trees for $i = \{1, ..., k\}$. The cardinality of τ_i may be exponentially large for each multicast request. Let $v_i(T)$ be a decision variable, such that $v_i(T) = 1$ if $T \in \tau_i$ is chosen to span multicast request S_i, and 0 otherwise. Let λ be the maximum congestion in a Steiner tree packing. The objective is to minimize the maximum congestion on an edge. Then, the integer programming model for multicast congestion problem is as follows:

$$\begin{aligned}
min \quad & \lambda \\
s.t. \quad & \\
\sum_{T \in \tau_i} v_i(T) &= 1 \qquad \forall \;\; i \in \{1, ..., k\} \\
\sum_{i=1}^{k} \sum_{T \in \tau_i \& e \in T} v_i(T) &\leq \lambda \qquad \forall \;\; e \in E \\
v_i(T) &\in \{0, 1\} \qquad \forall \;\; i \;\; \& \;\; \forall \;\; T \in \tau_i
\end{aligned}$$

Although our problem is a special case of the multicast congestion problem, the above model cannot be directly implemented for our problem. Moreover, for the global routing problem and its variations, integer programming methods have been considered by Lengauer and Lügering [63, 64], and Grötschel et al. [40, 41, 42, 43, 44] and Chopra [15]. But, these formulations do not directly apply to the branch decomposition problem as well since the Steiner trees are restricted to be packed on a ternary tree in G'. We offer different formulations that are more convenient to the special structure of our problem. We present our models in the next section.

IV.4. Formulations

In this section, we present the first integer programming formulations for the branch-width problem. We construct our models benefitting from Theorem 1.

IV.4.1. Decision variables

Define z to be a decision variable corresponding to the largest order of any separation in a branch decomposition. We can also think of z being the largest load on an edge in a Steiner tree packing. Then the objective function is z, and our goal is to minimize z.

We introduce the decision variables u_{ij} for the edges $(i, j) \in A$. We define $u_{ij} = 1$ if in the branch decomposition there is an edge between the two Steiner nodes $i, j \in N$, 0 otherwise. For simplicity, u_{ji} for $j > i$ implies that we are actually referring to u_{ij}. We have a set of variables t_{ei} for the edges $(e, i) \in B$. The variable $t_{ei} = 1$, if a leaf node e in the branch decomposition, a node in the set M corresponding to an edge $e \in E$ in the original graph, is connected to Steiner node $i \in N$, and 0 otherwise. The notation for the edges in E and the nodes in M are used interchangeably, since there is a bijection between them. We define z^v_{ij} for each edge $(i, j) \in A$ and $v \in V$. Lets denote the Steiner tree that spans the nodes in Q_v by T_v. Then, $z^v_{ij} = 1$ if the edge $(i, j) \in A$ is employed in the Steiner tree T_v that spans the net Q_v, and 0 otherwise. In a Steiner tree that connects the nodes of Q_v, there exists a unique path between each pair of nodes. So, we introduce y^{ef}_{ij} for each $(i, j) \in A$ and $e, f \in Q_v$ for each $v \in V$, where $y^{ef}_{ij} = 1$, if the edge $(i, j) \in A$ is on the path between the leaf nodes e and f in the Steiner tree T_v, and as well as in the branch decomposition, for $e, f \in Q_v$ and $v \in V$, 0 otherwise. We don't need to deal with every pair of incident edges of a vertex, since every edge on a Steiner tree is on at least one of the paths between a leaf and the others. Also, we define q^{ef}_i for each $i \in N$ and $e, f \in Q_v$, where $q^{ef}_i = 1$, if the node i is on the path in between the leaves e and f in the Steiner tree T_v, and so in the branch decomposition, for $e, f \in Q_v$ and $v \in V$, 0 otherwise.

Another set of decision variables are x^v_i variables, where $x^v_i = 1$ if Steiner node i

is in Steiner tree T_v, and 0 otherwise. The variables y_{ij}^{ef} and q_i^{ef} are replaced with x_i^v variables for the second formulation.

IV.4.2. Constraints

First, we set the degree of the nodes in set M to 1, so that a node corresponding to an edge in the original graph has exactly one neighbor, which is a Steiner node. Let $|E| = m$. We need the following set of m constraints for this purpose:

$$\sum_{i \in N} t_{ei} = 1 \qquad \forall \;\; e \in E \tag{4.1}$$

Since the branch decomposition is a tree, then the subgraph obtained by deleting all the leaves of the branch decomposition tree is also a tree that consists of $|N|$ vertices. It has $|N| - 1$ edges. We ensure this by setting the sum of the u_{ij} variables over set A to $|N| - 1 = |E| - 3$.

$$\sum_{(i,j) \in A} u_{ij} = |E| - 3 \tag{4.2}$$

Also, we know that each Steiner node has degree 3 in the branch decomposition. We ensure this with a set of $|N| = |E| - 2 = m - 2$ constraints. The constraint set below sets the degree of each $i \in N$ to 3.

$$\sum_{(i,j) \in A} u_{ij} + \sum_{(e,i) \in B} t_{ei} = 3 \qquad \forall \;\; i \in N \tag{4.3}$$

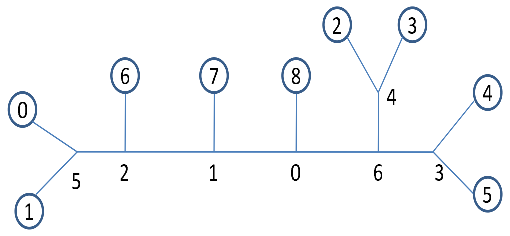
At most two of the 3 edges incident with a Steiner node in the branch decomposition can be edges from set B and at least one of them is an edge from set A. Thus, we have two set of $|N| = m - 2$ constraints, one of which is sufficient to be used in the model. These constraints are as follows:

$$\begin{aligned} \sum_{e \in M} t_{ei} &\leq 2 \qquad \forall \ i \in N \\ \sum_{j \in N} u_{ij} &\geq 1 \qquad \forall \ i \in N \end{aligned} \tag{4.4}$$

Fomin et al. [32] showed that given a 2-edge-connected graph G and a branch decomposition of G of width k, there also exists a connected branch decomposition (defined in Section II.2) of G of width $k' \leq k$. They presented a polynomial time algorithm that constructs a connected branch decomposition of width at most k, given a 2-edge-connected input graph G and a branch decomposition T of width k for G [32]. Thus, assuming the input graph is 2-edge-connected, if two edges in the input graph are not incident to each other, then there exists an optimal branch decomposition where the leaves corresponding to those two edges do not share a Steiner node. We add a constraint for each pair of non-incident edges in the original graph and this helps us narrow our feasible region by omitting some branch decompositions that are not connected. There are $\binom{m}{2}$ possible pairs of edges. Some of these pairs are connected. Thus, there are $O(m^2)$ of these constraints.

$$t_{ei} + t_{fi} \leq 1 \qquad \forall \ e, f \ \text{ s.t. } \ e, f \ \text{ don't share a vertex} \tag{4.5}$$

We reduce the symmetry by using a function $a(j) = 2^j + 1$ to narrow the feasible region even more. The label of the leaf (or the maximum label, if there are two leaves) connected to a Steiner node with a smaller index is always strictly greater than the label of the leaf (or the maximum label, if there are two leaves) connected to a Steiner node with a larger index. In other words, the labels of the Steiner nodes are ordered in the decreasing order of the maximum label of the leaves connected to a single Steiner node. In fact, we choose the function $a(j)$ in a fashion that there will be only a few ways of labeling for Steiner nodes of a certain branch decomposition. The number of ways to label Steiner nodes increase with the number of Steiner nodes connected to no leaves.



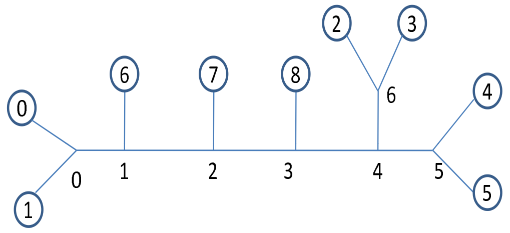


Fig. 8 : Two different labeling of the same branch decomposition

Since there are $m-2$ Steiner nodes in a branch decomposition, the anti-symmetry constraints will eliminate many of $(m-2)!$ feasible solutions for each branch decomposition. This saves a considerable amount of time. Figure 8 shows two branch decompositions that are same in essence, but the second one will be eliminated from the feasible solutions by means of the anti-symmetry constraints. We devise a set of $m-2$ constraints. These anti-symmetry constraints are as follows:

$$\sum_{e_j \in M} (w(j) * t_{e_j i} - w(j) * t_{e_j (i+1)}) \geq 0 \qquad \forall \;\; i \in \{0, 1..., |N|-1\} \tag{4.6}$$

Another set of constraints are based on the idea of a Steiner node being used in a Steiner tree on the path between two leaves. Let e_1 and e_2 be two edges in I_v

and let the leaves corresponding to these edges have the same name. There are 4 configurations;

1- Both (e_1, i) and (e_2, i) is in T_v,

2- Either (e_1, i) or (e_2, i) is in T_v, but not both,

3- Neither (e_1, i) nor (e_2, i) is in T_v, but i is in T_v, and

4- i is not in T_v.

In the first three cases the following holds:

$$t_{ei} + t_{fi} + \sum_{j \in (N - \{i\})} y_{ij}^{ef} = 2 \qquad \forall \;\; i \in N, \quad e, f \in Q_v, \quad v \in V$$

For the last case, the above constraint with the right hand side equal to 0 holds. We can combine these two constraints utilizing the decision variable we introduced as q_i^{ef}. Let $|V| = n$. We need a constraint for each combination of a Steiner node and two incident edges of a vertex in the input graph. There are $m - 2$ Steiner nodes. Assuming one edge is fixed in each pair of incident edges of a vertex, there are totally $2m - n$ pairs of incident edges. Then, we devise the following set of $(m-2)(2m-n)$ constraints to merge the above constraints:

$$t_{ei} + t_{fi} + \sum_{j \in (N - \{i\})} y_{ij}^{ef} - 2 * q_i^{ef} = 0 \qquad \forall \;\; i \in N, \quad e, f \in Q_v, \quad v \in V \quad (4.7)$$

This constraint set together with the constraint sets 4.1, 4.2 and 4.3 eliminate subtours and impose the connectivity requirement for the branch decomposition. This leads to a complete description of the associated polytope of the branchwidth problem.

Additionally, we have a set of constraints showing the ordering relationship between different sets of variables. For instance, y_{ij}^{ef} cannot be 1, if vertex i is not on the branch decomposition tree on the path from leaf e to leaf f. So, if $u_{ij} = 0$ for an edge (i,j) on the branch decomposition, then $y_{ij}^{ef} = 0$ for all pairs of e and f s.t $e, f \in Q_v$. The variable z_{ij}^{v} has to be equal to 1 if y_{ij}^{ef} is 1 for any $i,j \in N$, $e, f \in Q_v$ and $v \in V$. Because, if edge (i,j) is on the path between two leaves e and f, then it needs to be on the Steiner tree that employs e and f as leaves. Also, if edge (i,j) is on a Steiner tree then it needs to be on the branch decomposition. Hence, u_{ij} has to be equal to 1, if z_{ij}^{v} is 1 for any $i,j \in N$ and $v \in V$. If t_{ei} is 1, then at least one neighbor z_{ij}^{v} of i is 1 for any edge $e \in Q_v$. The reason for this is, every Steiner node in a Steiner tree has at least 1 edge that connects it to the other Steiner nodes. There are $\frac{(m-2)(m-3)(2m-n)}{2}$ constraints in the first set, $\frac{(m-2)(m-3)(2m-n)}{2}$ constraints in the second set, $\frac{n(m-2)(m-3)}{2}$ constraints in the third set, and $2m(m-2)$ in the fourth. These four set of constraints are as follows:

$$y_{ij}^{ef} \leq q_i^{ef} \qquad \forall \;\; i,j \in N, \quad e,f \in Q_v, \quad v \in V \tag{4.8}$$

$$y_{ij}^{ef} \leq z_{ij}^{v} \qquad \forall \;\; i,j \in N, \quad e,f \in Q_v, \quad v \in V \tag{4.9}$$

$$z_{ij}^{v} \leq u_{ij} \qquad \forall \;\; i,j \in N, \quad v \in V \tag{4.10}$$

$$t_{ei} \leq \sum_{j \in (N - \{i\})} z_{ij}^{v} \qquad \forall \;\; i \in N, \quad e \in Q_v, \quad v \in V \tag{4.11}$$

One last set of ordering constraints ensure that if the sum of y_{ij}^{ef} variables over (e,f) of edge pairs is 0, then u_{ij} is 0. Because, if edge (i,j) is not on the path between any pair of leaves, then it is not on the branch decomposition. There are $\frac{(m-2)(m-3)}{2}$ of the following constraints:

$$u_{ij} \leq \sum_{e,f \in E} y_{ij}^{ef} \qquad \forall \;\; i,j \in N \tag{4.12}$$

Moreover, we have a set of constraints for the Steiner trees. It is obvious that the number of edges between non-leaf nodes in a Steiner tree is not less than $|Q_v| - 2$, since these small trees also satisfy the properties of a tree and each has $|Q_v|$ leaves.

So, we have the following n constraints;

$$\sum_{(i,j)\in A} z_{ij}^v \quad \geq \quad |Q_v| - 2 \qquad \forall \ \ v \in V \tag{4.13}$$

A path satisfies the properties of a tree. Thus, the number of Steiner nodes in a path between two leaves on a Steiner tree is one more than the non-leaf edges on the path. We devise the following set of $\frac{(m-3)(2m-n)}{2}$ constraints:

$$\sum_{i\in N} q_i^{ef} - \sum_{(i,j)\in A} y_{ij}^{ef} \quad = \quad 1 \qquad \forall \ \ e, f \in Q_v, \quad v \in V \tag{4.14}$$

Lastly, we have a set of constraints for the cardinality of middle sets of the edges in the branch decomposition tree. The vertex v is in the middle set of the edge $(i,j) \in A$ in the branch decomposition, if and only if, $z_{ij}^v = 1$. Therefore, the cardinality of the middle set of edge $(i,j) \in A$ (or the load on edge (i,j)) is equal to the sum of z_{ij}^v variables over all the vertices in $|V|$. The variable z, is the largest cardinality of the middle sets, or the loads on edges. Thus, z is greater than or equal to the cardinality of all middle sets, or loads on edges. Then, we have the following $\frac{(m-2)(m-3)}{2}$ constraints:

$$\sum_{v\in V} z_{ij}^v \quad \leq \quad z \qquad \forall \ \ i,j \in N \tag{4.15}$$

Finally, our objective function is to minimize z. Below is the first formulation;

$$
\begin{array}{rcll}
min \quad z & & & \\
s.t. & & & \\
\sum_{i\in N} t_{ei} & = & 1 & \forall \;\; e \in M \\
\sum_{(i,j)\in A} u_{ij} & = & |E|-3 & \\
\sum_{(j\in N} u_{ij} + \sum_{e\in M} t_{ei} & = & 3 & \forall \;\; i \in N \\
\sum_{e\in M} t_{ei} & \leq & 2 & \forall \;\; i \in N \\
t_{ei} + t_{fi} & \leq & 1 & \forall\, i \in N,\; e, f \text{ s.t. } e, f \text{ don't share a vertex} \\
\sum_{e_j\in M} (a(j) * t_{e_j i} - a(j) * t_{e_j (i+1)}) & \geq & 0 & \forall \;\; i \in \{0, 1..., |N|\} \\
t_{ei} + t_{fi} + \sum_{j\in(N-\{i\})} y_{ij}^{ef} - 2 * q_i^{ef} & = & 0 & \forall \;\; i \in N, \quad e, f \in Q_v, \quad v \in V \\
y_{ij}^{ef} & \leq & q_i^{ef} & \forall \;\; i, j \in N, \quad e, f \in Q_v, \quad v \in V \\
y_{ij}^{ef} & \leq & z_{ij}^{v} & \forall \;\; i, j \in N, \quad e, f \in Q_v, \quad v \in V \\
z_{ij}^{v} & \leq & u_{ij} & \forall \;\; i, j \in N, \quad v \in V \\
t_{ei} & \leq & \sum_{j\in(N-\{i\})} z_{ij}^{v} & \forall \;\; i \in N, \quad e \in Q_v, \quad v \in V \\
u_{ij} & \leq & \sum_{e,f\in E} y_{ij}^{ef} & \forall \;\; i, j \in N \\
\sum_{(i,j)\in A} z_{ij}^{v} & \geq & |Q_v| - 2 & \forall \;\; v \in V \\
\sum_{i\in N} q_i^{ef} - \sum_{(i,j)\in A} y_{ij}^{ef} & = & 1 & \forall \;\; e, f \in Q_v, \quad v \in V \\
\sum_{v\in V} z_{ij}^{v} & \leq & z & \forall \;\; i, j \in N \\
u_{ij}, t_{ei}, y_{ij}^{ef}, q_i^{ef}, z_{ij}^{v} & \in & \{0, 1\} & \forall \;\; i, j \in N, \quad e, f \in Q_v, \quad v \in V \\
z & \in & \mathbb{Z} &
\end{array}
$$

In Model 1, there are totally $O(m^3)$ constraints. Table 1 shows the number of constraints in each constraint set for the first model.

For the second formulation, as we have mentioned before, we exclude the y_{ij}^{ef} and q_i^{ef} variables and add x_i^v variables to the model. As a reminder, $x_i^v = 1$, if Steiner node i is in Steiner tree T_v, and 0 otherwise. Related to this change in the variable set, we exclude some constraints and add new ones that have similar functions for the

Constraint set	# of constraints
(4.1)	m
(4.2)	1
(4.3)	$m-2$
(4.4)	$m-2$
(4.5)	$O(m^2)$
(4.6)	$m-2$
(4.7)	$(m-2)(2m-n)$
(4.8)	$\frac{(m-2)(m-3)(2m-n)}{2}$
(4.9)	$\frac{(m-2)(m-3)(2m-n)}{2}$
(4.10)	$\frac{n(m-2)(m-3)}{2}$
(4.11)	$2m(m-2)$
(4.12)	$\frac{(m-2)(m-3)}{2}$
(4.13)	n
(4.14)	$\frac{(m-3)(2m-n)}{2}$
(4.15)	$\frac{(m-2)(m-3)}{2}$

Table 1 : Number of constraints in Model 1

second formulation. Instead of constraint set 4.7, we add the following constraint set, which has a similar function to the omitted constraint. The below set of constraints ensure that if a Steiner node $i \in N$ is being used in a Steiner tree, then it is connected to at least 2 and at most 3 edges in that Steiner tree. There are $(2m-n)(m-2)$ of these constraints.

$$\sum_{e \in Q_v} t_{ei} + \sum_{j \in (N - \{i\})} z_{ij}^v - 2 * x_i^v \leq 1 \qquad \forall \;\; i \in N, \;\; e \in Q_v, \;\; v \in V \quad (4.16)$$

Moreover, we take out the ordering constraints 4.8, 4.9, 4.12, which contain the omitted set of variables, y_{ij}^{ef} and q_i^{ef}, and then we add some ordering constraints associated with the new variable set x_i^v. For all incident edges (i, j) of i, z_{ij}^v needs to be 0 if x_i^v is 0. Also, t_{ei} needs to be 0 if x_i^v is 0, where e is incident to v in the original graph. We represent these relationships with the following two sets of constraints. There are $\frac{n(m-2)(m-3)}{2}$ constraints in the first set, and $2m(m-2)$ constraints in the second.

$$z_{ij}^v \leq x_i^v \qquad \forall \;\; (i,j) \in A, \;\; v \in V \quad (4.17)$$

$$t_{ei} \leq x_i^v \qquad \forall \;\; (i) \in N, \;\; e \in Q_v, \;\; v \in V \quad (4.18)$$

One last set of constraints we add for the second formulation has a similar function to 4.14. Thus, we omit this set and add the following set of n constraints that ensures the number of vertices in a Steiner tree to be one more than the number of edges in that Steiner tree:

$$\sum_{i \in N} x_i^v - \sum_{(i,j) \in N} z_{ij}^v = 1 \qquad \forall \;\; v \in V \quad (4.19)$$

Finally, the rest of the second formulation is exactly same with the first formulation, including the objective function. There is an additional issue about the second formulation, which we will explain in Section IV.7. In Model 2, there are $O(nm^2)$ constraints. Table 2 shows the number of constraints in each constraint set for the second model.

Constraint set	# of constraints
(4.1)	m
(4.2)	1
(4.3)	$m-2$
(4.4)	$m-2$
(4.5)	$O(m^2)$
(4.6)	$m-2$
(4.16)	$(2m-n)(m-2)$
(4.17)	$\frac{n(m-2)(m-3)}{2}$
(4.18)	$2m(m-2)$
(4.10)	$\frac{n(m-2)(m-3)}{2}$
(4.11)	$2m(m-2)$
(4.13)	n
(4.19)	n
(4.15)	$\frac{(m-2)(m-3)}{2}$

Table 2 : Number of constraints in Model 2

IV.5. Upper and lower bounds

In this section, we explain the upper and lower bound heuristics that we apply to our model. These bounds, especially the lower bounds, are very important for this problem. Good bounds can dramatically decrease the run time. Since the problem is a minimization problem, a simple feasible solution obtained by a heuristic can give us an upper bound. So, it is not very difficult to find upper bounds. The reason why lower bounds are more important is that, since this is a minimization problem, any lower bound close to the optimal solution would greatly narrow the feasible region, and decrease the run time of the branch and bound algorithm. Also, in our experiments, we observed that we need good lower bounds to make the model perform faster. Otherwise, it takes a considerable amount of time for the branch and bound procedure to search for the different values of the variables, as illustrated in Table 3 and 4.

Tables 3 and 4 serve as a basis of comparison for the later results of Section IV.9. We run our models using C++ and CPLEX 9.0 on a Pentium 4 computer with 3.06 GHz CPU. In both tables, the times are given in seconds. An "X" on any of the tables means that the model does not terminate before 18000 seconds. We illustrate the run times for some selected test instances of the table on page 54. The title "β" stands for branchwidth. The column titled "run" in Table 3 shows the run time when no cut is applied to the first model. For many of these test instances upper bound is 3, which means there is no sufficient space for the cuts to be effective. The model will search for branch decompositions with width 2 only since the upper bound is 3 and the lower bound is 2 for biconnected graphs. The column titled "run" in Table 4 shows the run time when all cuts are applied to the second model. Some of the cuts are required for this model to be complete, that is why we record the run time with the cuts. The column titled "run" in Table 3 shows the run time when all cuts are applied to the first model to have a conformable comparison. The cuts will be explained later in Section IV.7.

If we do not apply any lower bound heuristic, Model 1 has run time greater than 18,000 seconds for 10 test instances, however Model 2 can solve 4 of these instances in a short time. If we compare the run times, it is clear that the second model performs better when no lower bound heuristic is used.

Graph	$(\lvert V\rvert, \lvert E\rvert)$	β	run
$W_{6,2}$	(6,9)	3	6.1
K_5	(5,10)	4	X
$W_{8,3}$	(8,12)	4	X
sudtbl	(11,14)	3	0.1
K_6	(6,15)	4	X
$W_{10,4}$	(10,15)	4	X
Petersen	(10,15)	4	X
$W_{12,5}$	(12,18)	4	X
laspow	(14,18)	3	140.1
nprio	(17,22)	3	1.3
drigl	(21,29)	3	0
si	(23,31)	3	2.7
seval	(24,31)	3	0.4
dyeh	(27,35)	3	2.7
linj	(27,35)	3	0.6
recre	(27,35)	3	X
bilan	(37,49)	3	413.2
tcomp	(38,50)	3	X
colbur	(40,54)	3	300
sortie	(44,62)	3	X
cardeb	(48,64)	3	X
yeh	(50,68)	3	207.9
pastem	(158,214)	3	NA

Table 3 : Model 1 run times without the lower bound heuristic

Graph	$(\lvert V\rvert, \lvert E\rvert)$	β	run
$W_{6,2}$	(6,9)	3	5.4
K_5	(5,10)	4	X
$W_{8,3}$	(8,12)	4	X
sudtbl	(11,14)	3	0.1
K_6	(6,15)	4	X
$W_{10,4}$	(10,15)	4	X
Petersen	(10,15)	4	X
$W_{12,5}$	(12,18)	4	X
laspow	(14,18)	3	0.1
nprio	(17,22)	3	0.1
drigl	(21,29)	3	0.2
si	(23,31)	3	0.3
seval	(24,31)	3	0.3
dyeh	(27,35)	3	0.3
linj	(27,35)	3	0.3
recre	(27,35)	3	0.3
bilan	(37,49)	3	0.7
tcomp	(38,50)	3	1.5
colbur	(40,54)	3	69.2
sortie	(44,62)	3	1.7
cardeb	(48,64)	3	15.8
yeh	(50,68)	3	67.4
pastem	(158,214)	3	NA

Table 4 : Model 2 run times without the lower bound heuristic

IV.5.1. Upper bound heuristics

Cook and Seymour [16] introduced a heuristic for finding good branch decompositions of a general graph. Then, Hicks [48] implemented two other heuristics, called diameter and hybrid methods, that perform better experimentally. Both methods are composed of a tree building and a separation finding component. In our model, we use the hybrid method and add the width obtained using this heuristic as an upper bound for the objective function.

We present a heuristic that we have devised for finding good branch decompositions. However, it does not perform as well as the above heuristics and is not used in our model. This upper bound heuristic constructs a simple structured branch decomposition with two Steiner nodes connected to two leaves and the rest connected to one leaf. This structure of width k in essence is equivalent to so called *monotone k-expansion* defined by Fomin et al. in [32]. It has been shown that monotone expansion number, i.e., the minimum k for which there is a monotone k-expansion, is an upper bound for branchwidth [32]. For detailed information on k-expansions the reader is referred to [32].

Briefly, the algorithm is as follows. Given a graph G, start with a minimum degree vertex v of G. Select two incident edges of v and connect them to a Steiner node in order to start forming the branch decomposition. Label these edges and their ends in G. If there is an unlabeled edge with both ends labeled, add that edge to the branch decomposition connecting it to a Steiner node, and then connecting this Steiner node to the last Steiner node we have added to the branch decomposition. If there is no such edge, then find a labeled vertex with minimum number of unlabeled incident edges. Add one of the incident edges to the branch decomposition using the same method as in the first case, and label this edge and its unlabeled ends in G.

Repeat the last steps until there are only two unlabeled edges. Then, connect these last two edges to a Steiner node and connect this node to the last Steiner node we have added. This algorithm gives a branch decomposition tree of a simple structure, where all the Steiner nodes are connected to at least one leaf. Thus, there are no Steiner nodes that are incident to none of the leaves. The bounds generated by this heuristic cannot beat the bounds of the previously mentioned three methods. We used the hybrid method to obtain upper bounds for our models.

Step 1: At each branch node, find the LP relaxation values for the edges between each pair of Steiner nodes (u_{ij} variables)

Step 2: Construct a graph of Steiner nodes and the edges connecting them

Step 3: Assign the LP relaxation values as the edge weights

Step 4: Find a spanning ternary tree on the graph formed of Steiner nodes

Step 5: Find the LP relaxation values for the edges between a leaf node and a Steiner node (t_{ei} variables)

Step 6: Recursively, find the largest LP relaxation value and, if possible, assign the corresponding leaf node to the associated Steiner node (ensuring that the properties of branch decomposition are not violated)

Step 7: Find the width of the resulting branch decomposition

Fig. 9 : Upper bound heuristic using LP relaxation values

In addition, we use a second upper bound heuristic, which we have devised. This heuristic is based on a spanning ternary tree constructed using the LP relaxation values of u_{ij} and t_{ei} variables at each branch node in the branch and bound tree. The heuristic is described in Figure 9. If the width found at the current branch node using this heuristic is lower than the best known upper bound, then the upper bound is updated and a cut representing the new upper bound is added to the model.

IV.5.2. Lower bound heuristic

The treewidth and branchwidth of a graph bound each other by constants. It has been proved that for a graph $G = (V, E)$ $E \neq \emptyset$, $max(\beta(G), 2) \leq \tau(G) + 1 \leq max(\lfloor \frac{3}{2}\beta(G) \rfloor, 2)$, where $\tau(G)$ is the treewidth of G [79]. Thus, any lower bound for treewidth leads to a lower bound for branchwidth of a graph. The lower bound heuristic we used is based on a treewidth lower bound offered by Bodlaender et al. [8], which is called contraction degeneracy. There are other lower bounds presented by [8] as well. But, in terms of the performance of the heuristics presented for these lower bounds, one of the contraction degeneracy heuristics stands out. It runs in a reasonable amount of time, and gives good lower bounds. This is the reason why we used a contraction degeneracy heuristic.

The *contraction degeneracy* of a graph G is defined to be the maximum over all minors of G of the minimum degree. So, $\delta C(G) := max_{G'}\{G'$ is a minor of G and $|V| \geq 1\}$. To determine if the contraction degeneracy of a graph G is at least k for some integer k is NP-Complete [8]. Bodlaender et al. [8] implemented several heuristics with different contraction strategies to find lower bounds for the contraction degeneracy. The one that has been shown to be the best performing experimentally in general was used for our computational results. Given an input graph $G = (V, E)$, the contraction degeneracy heuristic is summarized in Figure 10. The term "mind(G)" represents the minimum degree of graph G.

```
Step 1: Make a copy $G'$ of $G$, set the contraction degeneracy to be
        mind($G'$).
Step 2: Find a minimum degree vertex $v$ of $G'$ randomly.
Step 3: Select a neighbor vertex $u$ of $v$, such that $u$ and $v$ have the least
        number of common neighbors.
Step 4: Contract edge $(u, v)$, update $G'$, and compute mind($G'$).
Step 5: If mind($G'$) is larger than the contraction degeneracy, update
        contraction degeneracy.
Step 6: Repeat steps 2-5 until $G'$ has less than 4 vertices.
Step 7: Repeat steps 1-6 at most $\frac{|V|}{mind(G)}$ many times, and select the
        obtained highest contraction degeneracy
```

Fig. 10 : Contraction degeneracy heuristic

Usually in Step 2 of this algorithm, we encounter ties. Bodlaender et al. [8] do not mention a method to break the ties. In our implementation we break the ties randomly. [8] mentions that this heuristic may not perform very well for sparse graphs, but we obtained good lower bounds for sparse graphs in our experiments, which we think might be a consequence of random selection of minimum degree vertices, and repeating this randomized algorithm for many times to get the best bound.

We compute a good contraction degeneracy lower bound, $\delta_{LB}C(G)$, using this heuristic. Since contraction degeneracy is a lower bound for treewidth and treewidth is a lower bound of branchwidth, we have the following relationship:

$$\begin{aligned} \delta_{LB}C(G) + 1 \leq \tau(G) + 1 &\leq max(\lfloor \frac{3}{2}\beta(G) \rfloor, 2) \\ \Rightarrow \lceil \frac{2}{3}(\delta_{LB}C(G) + 1) \rceil &\leq \beta(G) \end{aligned}$$

We add the value of $\lceil \frac{2}{3}(\delta_{LB}C(G) + 1) \rceil$ as a lower bound for the variable z, and thus for the objective function in our models.

IV.6. Preprocessing

The leaves of a branch decomposition corresponding to the edges incident with vertices of degree one and two can be assigned to the branch decomposition tree in a way that we can still obtain a branch decomposition of G of width no more than the branchwidth of G. Let v be a degree one vertex, and (v, u) be the edge incident with v. In an optimal branch decomposition, the leaf corresponding to edge (v, u) can be connected to the same Steiner node with the leaf corresponding to one of the incident edges of u. This assignment will not increase the width of the branch decomposition. In other words, there exists an optimal branch decomposition of G such that the leaf corresponding to edge (v, u) is connected to a Steiner node where the leaf corresponding to one of the incident edges of u, other than (v, u) is connected to. Since we assumed our input graph to be a biconnected graph, there are no degree one vertices. However, this reduction can be useful if we sacrifice some of the benefits we obtained by having a biconnected graph as an input, and use our models for graphs that are not biconnected.

In a biconnected graph, every separation of order 2 can be represented in an optimal branch decomposition, if it does not violate other determined separations [17]. Thus, we focus on the reductions on degree two vertices. The separation generated by the incident edges of a degree two vertex has order 2. The reduction on these vertices decrease the number of variables in the model. Similar to the case of degree one vertices, we can say that there exists an optimal branch decomposition of G such that the leaves corresponding to the incident edges of a degree two vertex are connected to the same Steiner node.

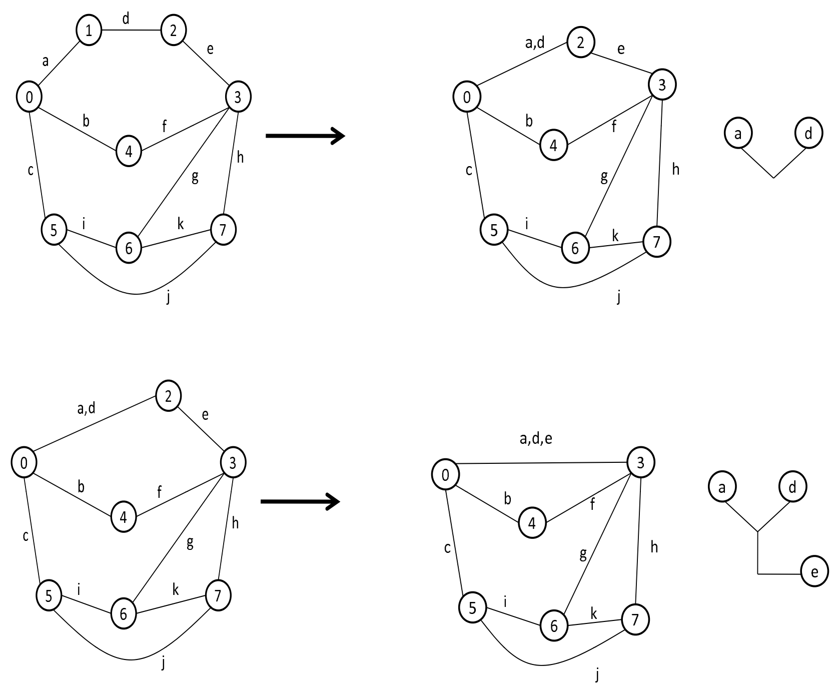


Fig. 11 : A series of reductions on G

In the reduction procedure, we first make a copy $\bar{G}$ of G. At each step we search for a vertex of degree two, do a reduction on this vertex and update $\bar{G}$. We repeat this procedure until we have no degree two vertices in $\bar{G}$. Let w be a vertex of degree two, and let $e = (w, q)$ and $f = (w, r)$ be the edges incident with w in $\bar{G}$. We can contract one of the edges e and f in $\bar{G}$, and connect the nodes in the branch decomposition corresponding to both edges to a designated Steiner node. Sometimes, an edge in $\bar{G}$ represents a subgraph of G, where the subgraph shrinks to an edge after a series of contractions. Hence, one or both of e and f may correspond to a set of edges that have already gone through the reduction process. In this case, the subtree corresponding to both set of edges is connected to a designated Steiner node. Figure 11 shows an example graph and a few steps of reduction process and branch decomposition tree construction.

IV.7. Cuts

Upper bound and lower bound heuristics and the preprocessing techniques are all efforts for decreasing the size of the feasible region, and so the running time of the algorithm. Another important method to decrease the running time is to add global or local cuts to the models. We implement four types of global cuts and one type of local cut.

The variable z represents the branchwidth of the input graph and also the maximum congestion on an edge, so it needs to be an integer. At every branch node, CPLEX finds an LP relaxation value for the variable z, then we cut off the solutions that employ values of z less than $\lceil z \rceil$, since we cannot get a value better than $\lceil z \rceil$ in the partial branch and bound tree that is rooted in the current branch node. Thus, at every branch node we add the following local cut;

$$z \geq \lceil z \rceil$$

One type of global cut is obtained as a result of the upper bound heuristic we have mentioned in Figure 9. We explain the other types of global cuts in the following section.

IV.7.1. γ, δ and δ-Steiner cuts

Let $T = (V_T, E_T)$ be a tree, and S be any subset of the vertex set V_T. Then, the number of edges where both ends lie in S is no larger than $|S| - 1$ for all $S \subseteq V_T$. This condition avoids subtours.

The graph formed by the u_{ij} variables is supposed to be a tree, which is a subgraph of the branch decomposition tree. Let us denote this tree by $T_U = (N, E_U)$. It is the induced subgraph of the branch decomposition tree formed by the Steiner nodes. It is supposed to satisfy the above mentioned condition. Thus, we can add a new cut to our model whenever we find a branch node where this condition is violated by the LP relaxation values of the u_{ij} variables. We use an algorithm by Cunningham [22] to determine some of the subsets of N that violates the above condition. Let R be one of the subsets of N such that the sum of the LP relaxation values of the edges where both ends lie in R is larger than $|R| - 1$. Then, we add the following cut, which is a subtour elimination constraint.

$$\sum_{i,j \in R} u_{ij} \quad \leq \quad |R| - 1$$

Let T be a tree and S be a subset of the vertex set of T. Then, the cardinality of the cut $(S, V_T - S)$ cannot be less than one. This constraint imposes the connectivity requirement.

Since T_U is a tree, it is supposed to satisfy the above condition. We use a minimum cut algorithm [74] to find the minimum cut in the graph formed by the LP relaxation values of the u_{ij} variables. If the value of the cut is less than one, then we add a new cut to the model. Let P be one of the subsets of N such that $(P, V_T - P)$ is a minimum cut and the cardinality of the cut is less than one. Then, we add the following cut;

$$\sum_{i\in P, j\in(V_T-P)} u_{ij} \;\geq\; 1$$

We obtain the δ-Steiner cuts using subtour elimination constraints on Steiner trees that span each terminal set. Since every Steiner tree is a tree, the cardinality of every cut on a Steiner tree needs to be at least one. We use a minimum cut on the graph formed of the LP relaxation values of the z_{ij}^v and t_{ei} variables. Let $G_v = (V_v, E_v)$ be the graph formed of the LP relaxation values of the z_{ij}^v and t_{ei} variables for a vertex v in V and all edges $e \in Q_v$. Recall that $G = (V, E)$ is the original input graph. Among all the subsets W of V_v where both $|W \cap Q_v| \geq 1$ and $|(V_v - W) \cap Q_v| \geq 1$ hold, let $(W', V_v - W')$ be the minimum cut with cardinality less than one. Then, we can add a cut to the model. One important issue we need to be careful about is that we cannot guarantee the Steiner nodes used for that Steiner tree at the current branch and bound node are the nodes that will be used for that Steiner tree in the optimal branch decomposition. That is why we use the set V instead of V_v in the following cut;

$$\sum_{i\in W', j\in(V-W')} z_{ij}^v + \sum_{e\in(W'\cap Q_v), i\in(V-W')} t_{ei} + \sum_{e\in(V_v-W')\cap Q_v, i\in(W'-Q_v)} t_{ei} \;\geq\; 1$$

These cuts correspond to the Steiner cut inequalities of Grötschel et al. [44]. We apply all the above mentioned cuts to both Model 1 and Model 2. For the second formulation, there is a subtle point that requires δ-Steiner constraints. Since we do not have path variables q_{ij}^{ef}, we cannot force the connectedness to the Steiner trees and to the branch decomposition tree. We need to have δ-Steiner inequalities as constraints in the second model to ensure the connectedness of the Steiner trees. Because of the large number of δ-Steiner constraints, instead of adding them to the model directly, we add them as cuts when they are violated.

IV.8. Formulation on hypergraphs

We implemented both models on hypergraphs. We can use the upper bound heuristic we developed on LP relaxation values directly. This heuristic is explained in detail in section IV.5.1. To be able to use the hybrid method as an upper bound and the contraction degeneracy heuristic as a lower bound we take advantage of the fact that branchwidth of a hypergraph is equal to the branchwidth of a graph constructed by using this hypergraph [81].

Let H be a hypergraph, and G be a graph constructed by replacing each hyperedge of form $\{u_1, ..., u_t\}$ with a complete bipartite graph with vertex set $\{u_1, ..., u_t, v_1, ..., v_t\}$ s.t. there is an edge connecting every u_i and v_j, $\{i, j\} \in \{1, ..., t\}$. Seymour and Thomas [81] proved that the branchwidth of H is equal to the branchwidth of G.

We cannot guarantee that the preprocessing steps we implemented on graphs are appropriate for hypergraphs. However, we do apply the cuts we used for graphs since these cuts are generated on the graphs formed by the LP relaxation values of the variables.

Note that the result on connected separations can be generalized to hypergraphs. Fomin et al. [32] presented a polynomial time algorithm that constructs a connected branch decomposition of width at most k, given a 2-edge-connected input graph G and a branch decomposition T of width k for G [32]. This algorithm can be directly used for branch decomposition of hypergraphs to obtain a connected branch decomposition of width at most k given a branch decomposition of width at most k, assuming that the hypergraph is 2-edge-connected. The proof in [32] is valid for hypergraphs as well. Hence, we can say that given a 2-edge-connected hypergraph H, there exists a connected branch decomposition of H of width equal to its branchwidth. Thus, we can use the constraint set (4.5) that forces two non-incident edges to be connected to different Steiner nodes in the branch decomposition.

IV.9. Computational results

We run our models using C++ and CPLEX 9.0. CPLEX has several parameters that are set to a default value and can be changed by the user. These parameters describe some details of the methods that will be used to obtain the solution, such as which variable to branch on first, which method to use for solving subproblems of a mixed integer program, etc. After considering the characteristics of our problem and obtaining some experimental results on different values of some parameters, we decided to keep most of the parameters in their default value, and set some other parameters as follows. The parameter ”FinalFactor” is described as the indicator for basis final factorization after the reverse operation (that occurs to build the original model when presolve changes the model). This parameter is set to ”on” as default, but we turned it ”off”. It is recommended for large models to save some time. The parameter ”Probe” determines the probing level on the variables before branching. In our experiments we tried all three levels (1-3)of probing and we set this parameter to 2. The parameter ”VarSel” represents the rule for selecting the branching variable at the branch node. The method we chose for ”VarSel” is strong branching. This method partially solves a number of subproblems with tentative branches and then decides to branch on the one that is the most promising. In terms of the cuts that CPLEX generates by default, we used the default aggressiveness level where CPLEX decides how aggressive it will apply a certain cut. The cuts that we generate in our model is used additional to the default cuts of CPLEX.

IV.9.1. Results on graphs

Our test instances are the biconnected graphs among compiler graphs used in [48], some well known graphs such as K_6, the Petersen graph, and the webs $W_{6,2}$, $W_{8,3}$, $W_{10,4}$, and $W_{12,5}$. Compiler graphs are the test instances of control-flow graphs from actual C compilations. These test instances were provided by Keith Cooper at Rice University. A web $W_{n,i}$ is a graph of n labeled vertices s.t. the edge set consists of the edges of the form $\{(j,(j+i)\text{Mod n}), ..., (j,(j-i)\text{Mod n})\}$. We also demonstrate run times of the graphs K_5, M_6, M_8 and Q_3 and a set of graphs we generate by deleting one edge from these graphs. The mentioned four graphs are *edge transitive*. Thus, deleting one edge is equivalent to deleting any other edge. We also demonstrate run times for some graphs we have generated and named as "g#_#". The branchwidth, upper and lower bounds for all of the test instances are shown in Table 5 and 6. The title "β" stands for the branchwidth. The columns titled "ub" and "lb" represent the upper and lower bounds for the branchwidth, respectively.

We use a Pentium 4 computer with 3.06 GHz CPU to obtain our results. The upper bound heuristic runs less than a second for all of the graphs and the lower bound heuristic runs less than 200 seconds except for a few of them. The upper bound and the lower bound heuristic run times are not shown in any of the tables for simplicity. For the graphs shown in Table 5 there is no gap between upper and lower bounds. These graphs are compiler graphs and 8 more graphs that we have mentioned before. Thus, both models terminate with the branch decomposition obtained by the upper bound heuristic.

Table 6 shows the test instances where there is a gap between upper and lower bounds. In all of these test instances, except $g28\_64$, the gap is only one. We set the upper bound of the branchwidth as 1 less than the width obtained from the hybrid

Graph	$(\|V\|,\|E\|)$	β	ub	lb
dens	(6,7)	2	2	2
$W_{6,2}$	(6,9)	3	3	3
K_5^-	(5,9)	2	2	2
K_5	(5,10)	4	4	4
cosqb	(8,10)	2	2	2
M_8^-	(8,11)	3	3	3
Q_3^-	(8,11)	3	3	3
M_6	(6,12)	4	4	4
coeray	(11,14)	2	2	2
sudtbl	(11,14)	3	3	3
K_6	(6,15)	4	4	4
arret	(12,15)	2	2	2
supp	(12,15)	2	2	2
$W_{12,5}$	(12,18)	4	4	4
laspow	(14,18)	3	3	3
tpart	(14,18)	2	2	2
inter	(16,21)	2	2	2
nprio	(17,22)	3	3	3
saturr	(17,23)	2	2	2
setinj	(18,23)	2	2	2
genb	(18,24)	2	2	2
laser	(19,25)	2	2	2
drigl	(21,29)	3	3	3
si	(23,31)	3	3	3
seval	(24,31)	3	3	3
bcndb	(25,34)	2	2	2
dyeh	(27,35)	3	3	3
linj	(27,35)	3	3	3
recre	(27,35)	3	3	3
injchk	(33,44)	2	2	2
bilan	(37,49)	3	3	3
tcomp	(38,50)	3	3	3
colbur	(40,54)	3	3	3
sortie	(44,62)	3	3	3
cardeb	(48,64)	3	3	3
prophy	(48,65)	2	2	2
yeh	(50,68)	3	3	3
verify	(55,73)	2	2	2
bcnd	(55,77)	2	2	2
tele56	(56,85)	3	3	3
heat	(69,95)	2	2	2
ddeflu	(92,124)	2	2	2
pastem	(158,214)	3	3	3

Table 5 : Branchwidth, upper and lower bounds without any gap

method. Thus, for these graphs the only thing the models need to verify is if the lower bound is feasible or not.

Graph	$(\|V\|, \|E\|)$	β	ub	lb
M_6^-	(6,11)	3	4	3
M_8	(8,12)	4	4	3
Q_3	(8,12)	4	4	3
$W_{8,3}$	(8,12)	4	4	3
$W_{10,4}$	(10,15)	4	4	3
Petersen	(10,15)	4	4	3
g11_23	(11,23)	5	5	4
g19_33	(19,31)	5	5	4
g14_40	(14,40)	6	6	5
g28_64	(28,64)	7	7	5

Table 6 : Branchwidth, upper and lower bounds with a gap

Table 7 and Table 8 show the building and running times for the test instances in Table 6 using the two formulations, respectively. Note that, when there is no gap between upper and lower bounds both models assume the branch decomposition given by the upper bound heuristic to be the optimal solution. Thus, building and running times for the graphs in Table 5 are less than a second. The columns titled "nc", "loc", "g&l", "d&l", "St&l", "all" show the run times when no cuts, only local cuts, local and γ cuts, local and δ cuts, local and δ-Steiner cuts, all types of cuts are applied, respectively. An "X" on any of the tables means that the model does not terminate before 18000 seconds.

If we compare run times of the test instances with the lower bound heuristic

Graph	$(\lvert V\rvert, \lvert E\rvert)$	β	build	nc	loc	g&l	d&l	St&l	all
M_6^-	(6,11)	3	1.4	327.1	103.8	116.8	354.5	234.9	37.3
M_8	(8,12)	4	2.9	X	X	X	X	X	X
Q_3	(8,12)	4	0.5	X	X	X	X	X	X
$W_{8,3}$	(8,12)	4	0.5	X	X	X	X	X	X
$W_{10,4}$	(10,15)	4	0.8	X	X	X	X	X	X
Petersen	(10,15)	4	0.6	X	X	X	X	X	X
$g11\_23$	(11,23)	5	0.9	X	X	X	X	X	X
$g19\_33$	(19,33)	5	1.7	X	X	X	X	X	X
$g14\_40$	(14,40)	5	3.1	X	X	X	X	X	X
$g28\_64$	(28,64)	7	12.8	X	X	X	X	X	X

Table 7 : Model 1 building and running times

with Table 3 and 4, we can see how useful the lower bound is. When the lower bound heuristic is applied the models terminate in less than a second for many of the graphs, since the lower bounds happen to be equal to the upper bounds. For three of the graphs it takes more than 18,000 seconds for both models to terminate without the lower bound heuristic. However, the lower bound heuristic can obtain a bound equal to the upper bound for these graphs.

As can be seen from Tables 7 and 8 the second model builds faster for M_6^-. However, Model 1 runs faster when all cuts are applied. Both models run for more than 18,000 seconds for M_8, Q_3, $W_{8,3}$, $W_{10,4}$ and Peterson graph. The second model terminates in a reasonable amount of time for the rest of the graphs, although the first one runs for more than 18,000 seconds. This might be because of the difference in the number of variables and the number of constraints in the two model. As a reminder, the models have $O(m^3)$ and $O(nm^2)$ constraints respectively. Also, we don't have y_{ij}^{ef} and q_i^{ef} variables in the second model, instead we have x_i^v variables. There is a huge decrease in the size of the variable set when we construct the second model starting with the first one since we don't need to consider all neighbor edge pairs and associated variables for the second model. Thus, in the branch and bound

Graph	$(\|V\|, \|E\|)$	β	build	run
M_6^-	(6,11)	3	0.5	70.9
M_8	(8,12)	4	0.6	X
Q_3	(8,12)	4	0.6	X
$W_{8,3}$	(8,12)	4	0.6	X
$W_{10,4}$	(10,15)	4	0.6	X
Petersen	(10,15)	4	0.6	X
$g11\_23$	(11,23)	5	0.9	19.5
$g19\_33$	(19,33)	5	2.8	70.8
$g14\_40$	(14,40)	5	2.2	144.3
$g28\_64$	(28,64)	7	10.7	9909.1

Table 8 : Model 2 building and running times

procedure, Model 2 has fewer variables to be branched on and fewer constraints of which feasibility check needs to be done. This might be the reason why Model 2 runs faster in general. In contrast Model 1, with all cuts, runs faster than Model 2 for one test instance. CPLEX runs some heuristics at some branch nodes to find feasible solutions to the problem to use as an upper bound. Model 1 with all cuts might have had a branch node where the heuristic comes up with a good feasible solution for that specific test instance.

In terms of comparison of the effectiveness of different cuts, we cannot come to a conclusion based on the test instances we have presented so far. For the graphs with less than 12 edges or for the compiler graphs that are very sparse and can be reduced to much smaller graphs with the help of reductions, the run times using different cuts do not show a great difference. For $W_{8,3}$, $W_{10,4}$, Petersen graph or larger graphs the run times are more than 18,000 seconds with both models and different combination of cuts. Thus, we cannot see if the cuts are effective, since we do not run them to completion. For this reason, we need small test instances that would run a reasonable amount of time and show the difference in the effectiveness of the cuts despite the fact that the functions that form the cuts take some time as well.

We use the graphs K_5^-, M_6^-, M_8^- and Q_3^- to show the effectiveness of the cuts. We do not use upper and lower bound heuristics to spare some space for the cuts to be effective. These graphs do not have many degree 2 vertices, which is a property we prefer for the purpose of decreasing the tractability of the input. Table 9 shows the run times of these test instances without any upper and lower bounds. We use a Pentium 4 computer with 3.06 GHz CPU to obtain our results. The mentioned graphs constitute the set that is obtained by deleting one edge from the obstruction set of graphs of branchwidth less than 4. All of these graphs have branchwidth 3 and adding the appropriate edge to each creates members of the aforementioned obstruction set, where all graphs have branchwidth 4. Thus, adding the necessary edge to our test instances increases the branchwidth by one. We think this might be the reason why the effectiveness of the cuts are better illustrated with these examples. Another reason might be the absence of the upper and lower bound heuristics. This leaves a larger feasible region for the models to run through, which also enables the cuts to be more effective.

As can be seen in Table 9, other than the last test instance, local and δ cuts perform the best. As a reminder, the columns titled “nc”, “loc”, “g&l”, “d&l”, “St&l”, “all” show the run times when no cuts, only local cuts, local and γ cuts, local and δ cuts, local and δ-Steiner cuts, all types of cuts are applied, respectively. Especially for “M_6^-” and “M_8^-” local and δ cuts together decrease the run time considerably, compared to the case when no cut is applied. The run times with

Graph	$(\lvert V\rvert, \lvert E\rvert)$	β	nc	loc	g&l	d&l	St&l	all
K_5^-	(5,9)	3	19.5	40.7	39.1	16.5	46.8	59.8
M_6^-	(6,11)	3	1399.2	1620.1	2075.4	394	3599.9	1056.8
M_8^-	(8,11)	3	1327	1739.2	1703.6	359.2	3656.8	1039.6
Q_3^-	(8,11)	3	23.7	44.9	46.3	62.4	34	70.7

Table 9 : Comparison of the efficiency of the cuts

the other cuts can be much larger than the run times with no cuts. This might be because these cuts are not as effective as the δ cuts, and the model wastes more time to generate these cuts, compared to the time it saves by using them.

IV.9.2. Results on hypergraphs

We use some test instances we have generated and referred to as hg#, where # represents numbers 1 through 8. Also, we use hypergraph representations of some circular clutters such as C_5^3, C_8^3, C_7^4, C_{11}^4, C_9^5 from [18]. The columns represent the nodes, and the rows represent the edges of the hypergraph. A few of the test instances are illustrated below:

$$C_5^3 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} \qquad C_7^4 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M(hg4)=\begin{pmatrix} 1&0&0&1&0&0&1&0&0&1&0&0&0&0&0&0\\ 0&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&1&1&1&1&0&0&0&1&1&0&0&0&0\\ 0&0&0&0&0&0&0&1&1&0&0&0&1&1&0&0\\ 1&1&0&0&0&0&0&0&1&1&0&0&0&1&1\\ 0&1&1&0&0&0&0&1&0&1&0&1&1&1&1&1\\ 0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&1\\ 0&0&0&0&1&0&1&0&1&0&1&0&1&0&0&0\\ 0&0&0&0&0&1&1&1&0&0&0&0&1&0&0&0 \end{pmatrix}$$

Table 10 shows the upper and lower bounds obtained for the hypergraph test instances. The title "β" stands for the branchwidth of the hypergraph. The columns titled "ub" and "lb" represent the upper and lower bounds for the branchwidth, respectively. An M in any of the tables means that the code runs out of memory for that test instance. An "X" on any of the tables means that the model does not terminate before 18000 seconds. The gap between upper and lower bounds mostly ranges from 0 to 3, however for the test instance $hg7$ the gap is 7. This might be because it is the graph that has the highest number of vertices over number of edges ratio among all of the test instances. Also, for all of the $C^{\#}_{\#}$ test instances, the above mentioned ratio is 1, and the gap between upper and lower bounds is at most 2.

Table 11 and 12 show the building and running times of Model 1&2, respectively, on hypergraphs. We use a Pentium 4 computer with 3.06 GHz CPU to obtain our results. Model 1 can handle hypergraphs with up to 8 or 9 hyperedges in a reasonable amount of time. For the larger hypergraphs, it takes more than 18,000 seconds to solve the problem to completion.

In terms of the effectiveness of the cuts for hypergraphs, local and δ cuts seem to perform better than the other cuts. The run times with local and δ cuts are shorter than the run times with other cuts (or without any cuts) for most of the test instances that terminate before 18,000 seconds.

Graph	$(\|V\|, \|E\|)$	β	ub	lb
hg1	(10,6)	6	7	6
hg2	(12,7)	8	9	7
hg3	(14,8)	8	9	7
hg4	(16,9)	10	10	8
hg5	(18,10)	10	11	8
hg6	(18,11)	11	11	8
hg7	(20,12)	NA	17	10
hg8	(18,13)	9	10	7
C_5^3	(5,5)	4	4	4
C_7^4	(7,7)	6	6	5
C_8^3	(8,8)	4	4	4
C_9^5	(9,9)	7	8	6
C_{11}^4	(11,11)	6	6	5
C_{11}^6	(11,11)	9	9	8
C_{13}^7	(13,13)	11	11	9

Table 10 : Branchwidth, upper and lower bounds for hypergraphs

Similar to the first model, Model 2 can handle graphs up to 8 or 9 hyperedges in a reasonable amount of time. The building and running times of Tables 11 and 12 are compared in Table 13. In general Model 2 builds and runs faster than Model 1 on hypergraphs, as in the case of graphs. However, for 8 of the test instances both models fail to terminate before 18,000 seconds, 6 of which Model 2 runs out of memory.

Graph	$(\|V\|, \|E\|)$	β	build	nc	loc	g&l	d&l	St&l	all
hg1	(10,6)	6	27.6	2.2	0.9	1.2	0.9	2.4	2.6
hg2	(12,7)	8	30.1	12	17.1	21	17.1	38.3	41.9
hg3	(14,8)	8	51.5	159	112	123.1	111.8	163.8	174.8
hg4	(16,9)	10	67.7	X	X	X	X	X	X
hg5	(18,10)	10	94.2	X	X	X	X	X	X
hg6	(18,11)	11	123	X	X	X	X	X	X
hg7	(20,12)	NA	162.7	X	X	X	X	X	X
hg8	(18,13)	9	128.5	X	X	X	X	X	X
C_5^3	(5,5)	4	1	0.2	0.3	0.3	0.3	0.4	0.4
C_7^4	(7,7)	6	1.2	12.5	19.2	23.4	19.2	43.5	47.6
C_8^3	(8,8)	4	1	2.2	3	3.6	3	4.2	4.9
C_9^5	(9,9)	7	41.8	4377.7	4828.63	4591.8	4462.2	6323.3	5831.7
C_{11}^4	(11,11)	6	57.7	X	X	X	X	X	X
C_{11}^6	(11,11)	9	138.55	X	X	X	X	X	X
C_{13}^7	(13,13)	11	332.8	X	X	X	X	X	X

Table 11 : Model 1 building and running times on hypergraphs

The results obtained for the hypergraphs are similar to the results obtained for the graphs in terms of the comparison of two models. The second model runs faster than the first model in general in graphs as well as in hypergraphs. As we have mentioned before, the reason might be the smaller size of the variable and the constraint sets of Model 2.

Table 14 shows the gap and the tree size obtained at the end of 5 hours for those hypergraphs that cannot reach the solution in less than 5 hours. The columns titled “gap1” and “gap2” show the gaps obtained by the two models, respectively. The gaps are shown as percentages. The columns titled “tree1” and “tree2” show the size of the trees obtained in the branch and bound procedure for the two models, respectively. The size of the trees are shown in megabytes.

Graph	$(\lvert V\rvert, \lvert E\rvert)$	β	build	run
hg1	(10,6)	6	0.5	2.1
hg2	(12,7)	8	0.5	15.1
hg3	(14,8)	8	0.6	115.9
hg4	(16,9)	10	0.7	M
hg5	(18,10)	10	1.1	M
hg6	(18,11)	11	1	M
hg7	(20,12)	NA	1	M
hg8	(18,13)	9	0.8	X
C_5^3	(5,5)	4	0.6	0.2
C_7^4	(7,7)	6	0.5	21.1
C_8^3	(8,8)	4	0.5	4.9
C_9^5	(9,9)	7	0.6	920.9
C_{11}^4	(11,11)	6	0.6	X
C_{11}^6	(11,11)	9	0.7	M
C_{13}^7	(13,13)	11	1.1	M

Table 12 : Model 2 building and running times on hypergraphs

Graph	$(\|V\|, \|E\|)$	β	build1	build2	run1	run2
hg1	(10,6)	6	27.6	0.5	2.6	2.1
hg2	(12,7)	8	30.1	0.5	41.9	15.1
hg3	(14,8)	8	51.5	0.6	174.8	115.9
hg4	(16,9)	10	67.7	0.7	X	M
hg5	(18,10)	10	94.2	1.1	X	M
hg6	(18,11)	11	123	1	X	M
hg7	(20,12)	NA	162.7	1	X	M
hg8	(18,13)	9	128.5	0.8	X	X
C_5^3	(5,5)	4	1	0.6	0.4	0.2
C_7^4	(7,7)	6	1.2	0.5	47.6	21.1
C_8^3	(8,8)	4	1	0.5	4.9	4.9
C_9^5	(9,9)	7	41.8	0.6	5831.7	920.9
C_{11}^4	(11,11)	6	57.7	0.6	X	X
C_{11}^6	(11,11)	9	138.55	0.7	X	M
C_{13}^7	(13,13)	11	332.8	1.1	X	M

Table 13 : Comparison of building and running times for hypergraphs

Graph	$(\lvert V\rvert, \lvert E\rvert)$	β	gap1	gap2	tree1	tree2
hg4	(16,9)	10	10%	10%	0.65	0.29
hg5	(18,10)	10	27.27%	27.27%	1.05	2.93
hg6	(18,11)	11	27.27%	27.27%	0.91	2.91
hg7	(20,12)	NA	33.33%	28.57%	0.56	1.78
hg8	(18,13)	9	30%	30%	0.61	1.99
C_{11}^4	(11,11)	6	16.67%	16.67%	0.51	0.25
C_{11}^6	(11,11)	9	11.11%	11.11%	0.83	0.83
C_{13}^7	(13,13)	11	18.18%	18.18%	1.05	0.73

Table 14 : Gaps and branch and bound tree size upon termination for hypergraphs

CHAPTER V

MODEL 3

V.1. Laminar separations and branch decompositions

We can think of a branch decomposition as an assignment of some separations of a graph G on the edges of a ternary tree structure, since every edge in a branch decomposition represents a separation. However, we need to do this assignment wisely so that the leaves of the branch decomposition tree corresponds to the separations where one part of the separation has cardinality one, and the maximum order of the separations represented in the branch decomposition is minimized. To build our third model, we utilize the relationship between the separations represented in a branch decomposition. Two separations (A_1, B_1) and (A_2, B_2) are called *laminar* if either $A_1 \subseteq A_2$, $A_1 \subseteq B_2$, $B_2 \subseteq A_1$, or $A_2 \subseteq A_1$. An edge e in a branch decomposition represents two separations (A_e, B_e) and (B_e, A_e), which are basically the same separations. In a branch decomposition the separation represented by an edge is laminar with all the separations represented by the other edges of the branch decomposition. In addition, if we have k pairwise laminar separations, they can all be represented by the edges of a single branch decomposition tree. Theorem 2 describes the relationship between laminar separations and branch decompositions.

Theorem 2. *If (T, ν) is a branch decomposition of G, the set of all separations of G represented by the edges of T is laminar. Conversely, if $(A_i, B_i) : 1 \leq i \leq k$ is a laminar set of separations of G, there is a branch decomposition (T, ν) such that:*

- *for $1 \leq i \leq k$, (A_i, B_i) is represented by a unique edge of T*
- *for each edge e of T, at least one of the separation represented by e equals (A_i, B_i) for some i, if k is equal to the total number of edges in T.*

Proof. First we prove that if (T, ν) is a branch decomposition of G, the set of all separations of G made by the edges of T is laminar. Let (A_1, B_1) and (A_2, B_2) be two separations of G, corresponding to two edges e_1 and e_2 of T respectively. Assume (A_1, B_1) and (A_2, B_2) are not laminar, then it is obvious that the intersection of the two sides of the separations are pairwise nonempty, i.e., $A_1 \cap A_2 \neq \emptyset$, $A_1 \cap B_2 \neq \emptyset$, $B_1 \cap A_2 \neq \emptyset$ and $B_1 \cap B_2 \neq \emptyset$. Let T_{A_1}, T_{B_1} and T_{A_2}, T_{B_2} be the subtrees obtained by removing e_1 and e_2 from T, respectively.

The intersection of subtrees of a tree is either empty set or a tree. Since the intersections are all assumed to be nonempty, then there exists a subtree of T corresponding to each intersection $A_1 \cap A_2$, $A_1 \cap B_2$, $B_1 \cap A_2$ and $B_1 \cap B_2$. We know that $A_1 \cup B_1 = A_2 \cup B_2 = E$. Thus, $(A_1 \cap A_2) \cup (A_1 \cap B_2) = A_1$, and we have the same result for the union of other intersections. Hence, in T there exists an edge connecting $T_{A_1 \cap A_2}$ to $T_{A_1 \cap B_2}$, $T_{A_1 \cap A_2}$ to $T_{B_1 \cap A_2}$, $T_{A_1 \cap B_2}$ to $T_{B_1 \cap B_2}$ and $T_{B_1 \cap A_2}$ to $T_{B_1 \cap B_2}$. So, T is 2-edge connected, which contradicts to T being a tree. Thus, the pairwise intersections cannot be all nonempty. It is obvious that they cannot be all empty either, since we have $A_1 \cup B_1 = A_2 \cup B_2 = E$. Hence, if (T, ν) is a branch decomposition of G, then the set of all separations of G made by the edges of T is laminar.

For the other direction, we present an algorithm that helps us in the proof. At the end of this algorithm we get a branch decomposition if $k = 2|E| - 3$. Let $(A_i, B_i) : 1 \leq i \leq k$ be a laminar set of separations of G. The algorithm is stated in Figure 12.

Step 1: Order the separations by the size of the $min(|A_i|, |B_i|)$ from highest to lowest.

Step 2: Start with a single node that represents all the edges in E, and get the first separation in the list.

Step 3: If there exists a node that carries elements from both sides of the separation, then split this node into two nodes, which represents the partitions of that node formed by the current separation, and assign an edge (that represents the current separation) between these two nodes.

Step 4: Else if there exists a node that carries all the elements of the one side of the separation, and if it is not connected to a dummy node then split this node into two nodes, one of which is a dummy node that represents an empty set. This dummy node is a connection of the new node to the rest of the tree. If the node is connected to a dummy node already, then leave the node as it is.

Step 5: Get the next separation from the list.

Step 6: Repeat Step 3, 4 and 5 until the end of the separations list.

Fig. 12 : Building branch decomposition given k laminar separations

Note that since the separations are laminar (the intersection of the two sides of the separations cannot be all pairwise nonempty, and they are not all pairwise empty either), each time we pick the next separation in the list there will be at most one node that has elements belonging to both sides of the separation. If there is one, this single node can be split by means of that separation. If there is no such node then there is a node that represents exactly all the elements in one side of the separation.

Then, this node is connected to the rest of the graph with an edge that represents the current separation. At the end of the algorithm we obtain a tree with k edges, since we had represented each separation with an edge. It is obvious that for $1 \leq i \leq k$, (A_i, B_i) is made by a unique edge of this tree. $\square$

V.2. Formulation

In this section we present our third formulation. We benefit from Theorem 2, the relationship between laminar separations and branch decompositions. All the separations represented in a branch decomposition are pairwise laminar. Since there are $2|E| - 3$ edges in a branch decomposition, we look for $2|E| - 3$ pairwise laminar separations such that the branch decomposition formed by these separations has the optimal width.

V.2.1. Decision variables

Let S be the set of all separations of the graph G. There are $2^{|E|-1} - 1$ separations in a graph. We know that there are $|E|$ separations with one edge in one side of the partition. Let this set be denoted by U. The separations in U are represented in all branch decompositions of $|G|$ as the leaves of the branch decomposition. In addition, we assume that G is 2-edge-connected. Using the result in Fomin et al. [32], we can focus on the separations where the both sides of the separation is connected, since these separations are sufficient to find an optimal branch decomposition. Let us call such separations as connected separations. Let S' be the set of connected separations that are in $(S - U)$. This reduction in the set of separations is going to be very helpful in the effectiveness of our model. For instance, in Petersen graph we have $2^{14} - 1 = 16383$ separations, and 3772 connected separations.

We introduce decision variable x_s for each separation $s \in S'$. We define $x_s = 1$ if s is represented in the branch decomposition tree, 0 else. We define z to be the decision variable corresponding to the largest order of any separation in a branch decomposition like in the previous models. We write $s \bowtie s'$ if two separations s and s' are laminar.

V.2.2. Constraints

Among the $2|E| - 3$ separations represented in any branch decomposition of G, $|E|$ of them are the leaves, which correspond to the separations in set U. Thus, we are searching for the rest of the necessary separations that leads to an optimal branch decomposition. We need the number of employed separations in the model to sum up to $|E| - 3$. We need a single constraint for this purpose.

$$\sum_{s \in S'} x_s = |E| - 3 \tag{5.1}$$

It is obvious that any separation is laminar with the separations in set U. In addition, the $|E| - 3$ separations we are looking for needs to be pairwise laminar. This is ensured by the following set of constraints:

$$(|E| - 4)x_s - \sum_{s' \in S' \text{ s.t. } s \bowtie s'} x_{s'} \leq 0 \qquad \forall \;\; s \in S' \tag{5.2}$$

Let $F(s)$ be the order of separation for separation s. Note that, the order of a separation is the width of the edge corresponding to that separation in a branch decomposition. Since, the width of a branch decomposition is the maximum width over all edges of the branch decomposition, we have the following constraint set;

$$F(s)x_s \leq z \qquad \forall \;\; s \in S'$$

Finally, our objective is to minimize z like in the previous formulations.

V.3. Upper and lower bounds

We use the hybrid heuristic by Hicks [48] described in Section IV.5.1. We find a feasible solution with this method, which gives an upper bound for the branchwidth. To save time and reduce the number of feasible solutions, we discard the separations with order greater than the upper bound and update the set S'. Thus, we can have even fewer variables, if we have a better upper bound. Hence, we can save considerable amount of time in the branch and bound process. The last upper bound heuristic described in Figure 9 is not applicable for this model since it is based on the LP relaxation values of the u_{ij} variables. As a lower bound, we use the same lower bound we use for the first two models, which is a contraction degeneracy heuristic. In addition, we utilize the same preprocessing methods we used for the first two models, and fix some of the variables to 1 by means of these reductions.

V.4. Formulation on hypergraphs

We implemented Model 3 also on hypergraphs. In terms of a bound on the branchwidth, like in the first two models, we construct a graph using the input hypergraph as described by Seymour and Thomas [81], and set the upper and lower bounds on the branchwidth of the input hypergraph as the upper and lower bounds on the branchwidth of the constructed graph.

V.5. Computational results

We run our model using C++ and CPLEX 9.0. Our test instances are the same instances we used for Models 1 and 2.

V.5.1. Results on graphs

Table 15 shows the performance of the third model on the test instances with gap between upper and lower bounds. The column title "β" stands for the branchwidth and the columns titled with "build" and "run" shows the building and the running times of the model. An "X" on any of the tables means that the model does not terminate before 18000 seconds. In general, Model 3 performs well in terms of run time. However, the building times can be significantly greater than the building times of the first two models due to the construction of all connected separations. As we can see in Table 15, it takes more than 18,000 seconds to build the model for those test instances with more than 30 edges.

Graph	$(\|V\|, \|E\|)$	β	build	run
M_6^-	(6,11)	3	1.1	0.4
M_8	(8,12)	4	1.3	0.7
Q_3	(8,12)	4	1.3	1.1
$W_{8,3}$	(8,12)	4	1	0.4
$W_{10,4}$	(10,15)	4	5.5	2.3
Petersen	(10,15)	4	5.3	2.2
$g11\_23$	(11,23)	5	1890.7	14720.9
$g19\_33$	(19,31)	5	X	NA
$g14\_40$	(14,40)	5	X	NA
$g28\_64$	(28,64)	7	X	NA

Table 15 : Model 3 building and running times

Table 16 demonstrates a comparison of the building and the running times of all three models, for the test instances of Table 15. We use a Pentium 4 computer with 3.06 GHz CPU to obtain our results. The columns titled “build1”, “build2”, and “build3” show the building times of all three models, respectively. The columns titled “run1”, “run2”, and “run3” show the run times of all three models, respectively. The column “run1” is the run time when all cuts are applied to Model 1.

Graph	$(\|V\|, \|E\|)$	β	build1	build2	build3	run1	run2	run3
M_6^-	(6,11)	3	1.4	0.5	1.1	37.3	70.9	0.4
M_8	(8,12)	4	2.9	0.6	1.3	X	X	0.7
Q_3	(8,12)	4	0.5	0.6	1.3	X	X	1.1
$W_{8,3}$	(8,12)	4	0.5	0.6	1	X	X	0.4
$W_{10,4}$	(10,15)	4	0.8	0.6	5.5	X	X	2.3
Petersen	(10,15)	4	0.6	0.6	5.3	X	X	2.2
g11_23	(11,23)	5	0.9	0.9	1890.7	X	19.5	14720.9
g19_33	(19,33)	5	1.7	2.8	X	X	70.8	NA
g14_40	(14,40)	5	3.1	2.2	X	X	144.3	NA
g28_64	(28,64)	7	12.8	10.7	X	X	9909.1	NA

Table 16 : Comparison of building and running times

For the graphs with branchwidth 5 or more the second model performs the best. It builds and runs in a reasonable amount of time for these graphs. For the graphs with branchwidth 4, Model 3 performs much better than the first two models. It solves these instances in less than 5 seconds. For the test instances with less than 15 edges, the third model performs the best in general.

V.5.2. Results on hypergraphs

We use the test instances of section IV.9.2. Table 17 shows the building and running times of Model 3 on hypergraphs. An "X" on any of the tables means that the model does not terminate before 18000 seconds. We use a Pentium 4 computer with 3.06 GHz CPU to obtain our results.

Graph	$(\|V\|, \|E\|)$	β	build	run
hg1	(10,6)	6	13.5	0.1
hg2	(12,7)	8	27.8	0.1
hg3	(14,8)	8	49.5	0.1
hg4	(16,9)	10	59.8	2.2
hg5	(18,10)	10	101.3	8.7
hg6	(18,11)	11	137.4	82.5
hg7	(20,12)	NA	264.9	X
hg8	(18,13)	9	577.8	7570.3
C_5^3	(5,5)	4	3.6	0.1
C_7^4	(7,7)	6	13.8	0.1
C_8^3	(8,8)	4	13.8	0.1
C_9^5	(9,9)	7	42.8	0.2
C_{11}^4	(11,11)	6	98.2	0.2
C_{11}^6	(11,11)	9	142.5	2.6
C_{13}^7	(13,13)	11	753.6	3305.5

Table 17 : Model 3 building and running times on hypergraphs

Model 3 can handle hypergraphs with up to 13 hyperedges. However for the test instance “hg7” it runs for more than 18,000 seconds and still cannot solve it, although it has 12 edges. The reason might be the larger size of the vertex set of this hypergraph, or the performance of the upper and lower bound heuristics. Because for this test instance there is a gap of 7 between upper and lower bounds. Upon termination in 18,000 seconds, the size of the branch and bound tree is 13.74MB, and the gap is 14.25% for this test instance. The best feasible solution it finds has a width of 14 and the obtained best lower bound in the branch and bound procedure is 12.

Graph	$(\|V\|, \|E\|)$	β	build1	build2	build3	run1	run2	run3
hg1	(10,6)	6	27.6	0.5	13.5	2.6	2.1	0.1
hg2	(12,7)	8	30.1	0.5	27.8	41.9	15.1	0.1
hg3	(14,8)	8	51.5	0.6	49.5	174.8	115.9	0.1
hg4	(16,9)	10	67.7	0.7	59.8	X	X	2.2
hg5	(18,10)	10	94.2	1.1	101.3	X	X	8.7
hg6	(18,11)	11	123	1	137.4	X	X	82.5
hg7	(20,12)	NA	162.7	1	264.9	X	X	X
hg8	(18,13)	9	128.5	0.8	577.8	X	X	7570.3
C_5^3	(5,5)	4	1	0.6	3.6	0.4	0.2	0.1
C_7^4	(7,7)	6	1.2	0.5	13.8	47.6	21.1	0.1
C_8^3	(8,8)	4	1	0.5	13.8	4.9	4.9	0.1
C_9^5	(9,9)	7	41.8	0.6	42.8	5831.7	920.9	0.1
C_{11}^4	(11,11)	6	57.7	0.6	98.2	X	X	0.2
C_{11}^6	(11,11)	9	138.55	0.7	142.5	X	X	2.6
C_{13}^7	(13,13)	11	332.8	1.1	753.6	X	X	3305.5

Table 18 : Comparison of building and running times for hypergraphs

Table 18 compares the building and running times of all three models on hypergraphs. It is clear that the third model performs the best for the hypergraph test instances, which might be because these hypergraphs have less than 14 hyperedges. Thus, Model 3 can build all the test instances in a reasonable amount of time, and with the help of upper bounds it can eliminate many separations and keep the size of the model as small as possible. Hence, this makes the third model advantageous over the other two models for hypergraphs up to 13 hyperedges.

CHAPTER VI

RANK-WIDTH PROBLEM

The rank-width and the rank decompositions are used as a tool for finding $f(k)$-expression for a graph with clique-width k, where f is a fixed function. Clique-width, like treewidth and branchwidth, is used for solving NP-hard problems. Many hard problems can be solved in polynomial time if modeled on graphs of bounded clique-width, given corresponding k-expression of the input graph. Oum and Seymour [72] proved that, even if the k-expression is not provided, clique-width can still be utilized for solving NP-hard problems. This was achieved by means of rank decompositions. Thus, finding good rank decompositions can be crucial for the effectiveness of the algorithms that utilize clique-width.

The rank-width problem is very much related with the branchwidth problem. Given a graph G, the rank decomposition of G is the branch decomposition of the cut-rank function of G, and the rank-width problem is to compute the rank-width and an optimal rank decomposition of a graph. Chapter III gives detailed definitions of rank-width and optimal rank decomposition.

VI.1. Vertex-minors

The vertex-minors relationship of graphs is a graph containment relation similar to graph minors. As we have mentioned in section III.2, there is a nice relationship between graph minors and branch decompositions. For a given integer k, the class of graphs with branchwidth at most k is a minor closed class, meaning that if G has branchwidth at most k, then any minor of G has branchwidth at most k as well [79]. Vertex-minor containment was introduced with the name l-reduction by Bouchet [11] and its connection with rank-width was developed by Oum [69] while searching for a similar relationship for clique-width. For a fixed k, the class of graphs with rank-width at most k is a vertex-minor closed class. Thus, if G has rank-width at most k, then any vertex-minor of G has rank-width at most k [69].

Most of the terminology we use in this Chapter is taken from Oum [69]. Let $G = (V, E)$ be a graph and v be a vertex in V. *Deleting* v from G means removing v and its incident edges from G. The graph obtained by deleting v is denoted by $G \setminus v$. The *local complementation* at v is to replace the edges of the subgraph obtained by the neighbors of v, by the edges of its complement. The graph obtained by local complementation at v to G is denoted by $G * v$. For an edge (u, v), *pivoting* is to do local complementation at u, then v and then u again. The graph obtained by pivoting (u, v) is denoted by $G \wedge (u, v)$, and $G \wedge (u, v) = G * u * v * u = G * v * u * v$. Oum [69] showed that local complementation or pivoting does not change the rank-width, which means a graph obtained by a series of local complementations or pivotings has the same rank-width with the original graph. But, the rank-width of a graph obtained by a deletion of a vertex can be either equal to or one less of the rank-width of the original graph [69]. An example of a local complementation is shown in Figure 13.

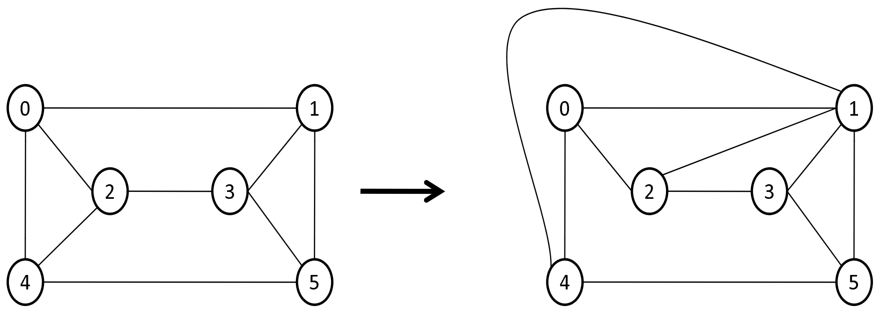


Fig. 13 : G and a local complementation at vertex 0

Figure 14 shows two graphs. The one on the left shows a graph obtained by local complementation at vertex 4 for the graph (on the right) of Figure 13 and the one on the right shows a local complementation at vertex 0 for the obtained new graph. The local complementations at vertices 0, 4 and 0 again is equivalent to pivoting (0,4) on the original graph (on the left) of Figure 13.

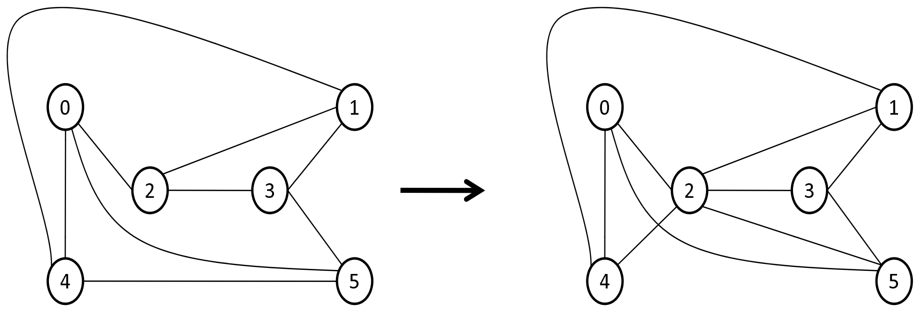


Fig. 14 : The graph obtained by local complementation at vertices 0 and 4 on G (left) and the graph obtained by pivoting (0,4) on G (right)

A graph H is a *vertex-minor* of G if H can be obtained by a series of vertex deletions and local complementations on G. H is a *pivot-minor* of G if H can be obtained by a series of vertex deletions and pivotings on G. If G has rank-width at most k, then any vertex-minor or pivot-minor of G has rank-width at most k [69].

VI.2. Formulation

The cut-rank function is defined on the vertex set of the input graph. Since the third model we have presented is based on the separations of a set, it can be used for rank-width problem as well. In a branch decomposition of a graph the separations of the edge set are employed, and the order of a separation is the number of vertices that have incident edges in both sides of the separation. But, for a rank decomposition we have the separations of the vertex set. Also, the order of a separation is defined as the rank of the submatrix of the adjacency matrix formed by the sides of the separation. Thus, we use the cut-rank function as the function F in the third model. Also, we apply some reductions on the input graph, which will be explained in detail in section VI.4.

VI.3. Upper and lower bounds

Oum [71] recently proved that the rank-width of a graph is less than or equal to its branchwidth. For this reason, any upper bound for the branchwidth can be an upper bound for the rank-width. We develop and implement a heuristic that computes a rank decomposition and the width of this decomposition. In our model we use the best bound obtained by the branchwidth heuristic (hybrid method) and the rank-width heuristic. Note that the bound obtained by the hybrid method does not provide a rank decomposition, but the rank-width heuristic does. Thus, if the hybrid method provides a better bound, then we eliminate the separations with cut-rank function value *greater than* this bound, but if the rank-width heuristic provides a better bound, then we eliminate the separations with cut-rank function value *greater than or equal to* this bound. In addition, with the help of preprocessing steps, we can check if the input graph has rank-width 1 or conclude that the lower bound for the rank-width is 2.

VI.3.1. The rank-width heuristic

The rank-width heuristic computes a rank decomposition and its width using a wise way of recursively determining the set of vertices that will be represented as a separation in the rank decomposition. We compare the width we obtain from this heuristic and the hybrid method, and the smaller one serves as an upper bound on the rank-width. Let G be the input graph. The rank-width heuristic is described in Figure 15.

Step 1: For each pair of nodes in G compute the value of the cut-rank function of the separation where one side consists of this pair of nodes.

Step 2: Among the node pairs of minimum cut-rank function value, select a pair that has minimum number of neighbors that are not in common.

Step 3: Identify the selected pair of nodes (the new node represents the two identified nodes), and update G.

Step 4: Connect the subtrees representing the identified nodes to a new non-leaf node. If the subtree is a single node connect it to the new non-leaf node by an edge. If the subtree is larger than a vertex, then connect the non-leaf node of degree 2 in this subtree to the new non-leaf node.

Step 5: Repeat Steps 1-4 until G has 3 nodes.

Step 6: Connect the three subtrees represented by the remaining 3 nodes to a new non-leaf node, as explained in Step 4.

Fig. 15 : Rank-width heuristic

Two vertices in a graph G are called *twins* if they are adjacent to exactly the same neighbor vertices in G. This algorithm naturally utilizes the degree one vertices and twins at the very beginning, since we can obtain a cut-rank function value of 1 using the degree one vertices or twins. The value of 1 is the minimum that can be achieved with a cut-rank function if the graph is connected. The algorithm searches for a pair of nodes that will result with a minimum cut-rank function value, and among such pairs selects a pair, which has minimum number of neighbors that are not in common. With this approach we try to keep the cut-rank function value of the selected separations as small as possible, while trying to have minimum number of columns in the submatrices of the selected separations, hoping that this might lead to separations of small cut-rank function value in the later steps. Thus, this is a type of greedy heuristic.

VI.3.2. Lower bound

Oum [69] proved that G is distance hereditary if and only if G has rank-width at most 1. A graph G is called *distance-hereditary* if and only if for every connected induced subgraph H of G, the distance between every pair of vertices in H is the same as in G. It is proved that every distance-hereditary graph can be obtained by creating twins and adding degree one vertices.

When we go through the preprocessing steps we contract the incident edges of degree one vertices and identify the twin vertices. The details of these reductions will be explained in the next section. At the end of this process if we end up with a single vertex, this means that the input graph is distance hereditary. Hence, the rank-width is 1. Otherwise, we set the lower bound of the rank-width for the input graph to 2.

VI.4. Preprocessing

In terms of preprocessing, we studied the reductions that could be done on vertices of degree one and two. Moreover, we apply a reduction on twin vertices.

VI.4.1. Degree one vertices and twins

We have a few observations on vertices of degree one and twins. Let $N(u)$ denote the neighbor set of a vertex u. Let the vertices u and v be twins. Then $N(u) - \{v\} = N(v) - \{u\}$. Thus, the value of the cut-rank function for the separation $\{u, v\}$ is 1. Then, it does not increase the width of the rank decomposition to have u and v connected to the same non-leaf node in the rank decomposition. Having one of these vertices connected to a non-leaf node is exactly the same with having both connected to the same non-leaf node, since the submatrices of the adjacency matrix formed by the separations $(\{u\}, (V \setminus \{u\})$, $(\{v\}, (V \setminus \{v\})$ and $(\{u, v\}, (V \setminus \{u, v\})$ are same after obtaining linearly independent rows.

For the reduction on the input graph, we identify the twin vertices and connect them to the same non-leaf node in the rank decomposition. Identifying twins is basically the same operation with deleting one of them. This leads us to a vertex minor of the original graph. In general, deleting a vertex, say u, may decrease the rank-width of the graph by one, but for this case the rank-width stays the same. We can obtain an optimal rank decomposition of the original graph by replacing v (assuming u and v are twins) on the optimal rank decomposition of the reduced graph by the pair u and v. The width does not increase with this replacement.

Now, let the vertex w be a vertex of degree one, and t be its neighbor. The value of the cut-rank function for the separation $\{w, t\}$ is 1, since t is the only neighbor of w. Then, it does not increase the width of the rank decomposition to have w and t connected to the same non-leaf node in the rank decomposition. Having vertex t connected to a non-leaf node is almost the same with having both t and w connected to a non-leaf node, since the submatrices of the adjacency matrix formed by the separations $(\{t\}, (V \setminus \{t\})$ and $(\{t, w\}, (V \setminus \{t, w\})$ has only one row that differs by only one column, which is the column represented by w.

In terms of the reduction on the input graph, we contract the incident edge of the vertex of degree one, and connect the degree one vertex and its neighbor to the same non-leaf node in the rank decomposition. Contracting the mentioned edge is the same operation with deleting the degree one vertex. This operation leads us to a vertex minor of the original graph. Like in the operation of identifying twins, the rank-width stays the same. We can obtain an optimal rank decomposition of the original graph by replacing t (assuming w is a degree one vertex and t is its only neighbor) on the optimal rank decomposition of the reduced graph by the pair t and w. The width does not increase with this replacement, since the submatrices of all the separations do not change.

VI.4.2. Degree two vertices

Let G be a graph and the vertices u and v be neighbors in G, and let both of them have degree two. So, u has exactly one neighbor other than v, and v has exactly one neighbor other than u. We claim that in an optimal rank decomposition u and v are connected to the same non-leaf node. To prove this we first prove the following theorem.

Theorem 3. *Given $G = (V, E)$, let u and v be two vertices of degree two and $(u, v) \in E$. For every separation $(A, V \setminus A)$ of V s.t. $\{u\} \subset A$ and $\{v\} \subset (V \setminus A)$, $\rho(A, V \setminus A) - 1 \leq \rho(A \cup \{v\}, V \setminus (A \cup \{v\})) \leq \rho(A, V \setminus A)$, where ρ represents the cut-rank function.*

Proof. This theorem states that every separation that separates u and v has a cut-rank function value that is same or exactly one more than the cut-rank function value of the separation obtained by carrying one of these two vertices to the other side of the separation.

Let $N(u) = \{v, w\}$ and $N(v) = \{u, t\}$. Let $(A, V \setminus A)$ be a separation of V s.t. $\{u\} \subset A$ and $\{v\} \subset (V \setminus A)$. Then there are three cases to consider. These cases are shown in Figure 16.

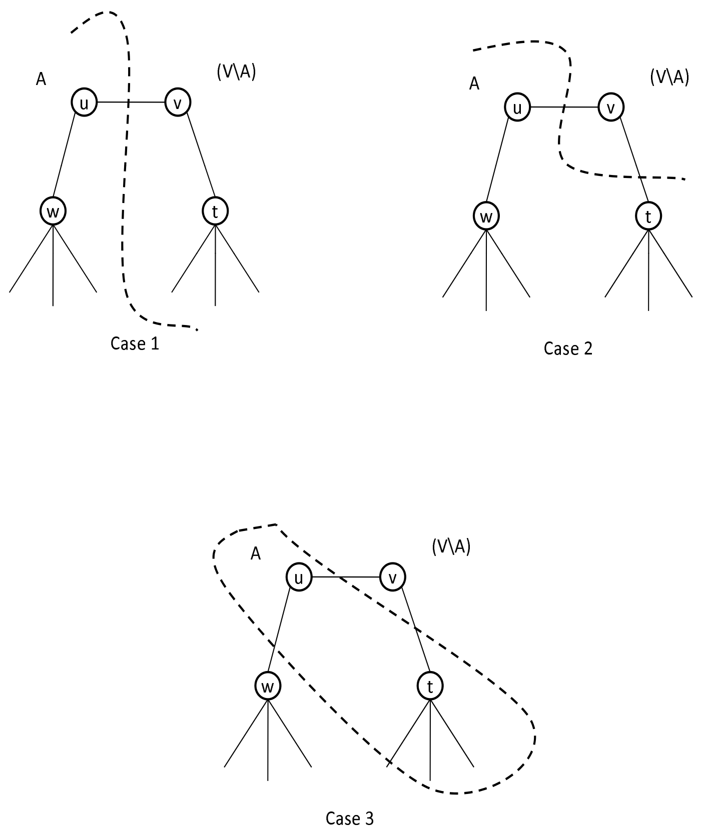


Fig. 16 : Three cases for set A

Case 1: If $w \in A$ and $t \in V \setminus A$, then the cut-rank function value of the new separation obtained by carrying v to the other side of the separation will be exactly the same with or one less than the cut-rank function value of the original separation. In the submatrix that represents $(A, V \setminus A)$ there is a row for vertex u that has a 1 on the column representing vertex v and 0 on other columns. In the submatrix that represents $(A \cup \{v\}, V \setminus (A \cup \{v\}))$, this row is replaced by a row that represents vertex v that has a 1 on the column representing vertex t and 0 elsewhere. Other rows stay the same. This new row may be linearly dependent to the other rows of the submatrix. Thus, the rank of this new matrix is either the same with or one less than the previous matrix.

Case 2: If $w, t \in A$, then the cut-rank function value of the new separation obtained by carrying v to the other side of the separation will be clearly one less than the cut-rank function value of the original separation, since all neighbors of vertex v is in A.

Case 3: If $t \in A$ and $w \in V \setminus A$, then similar to case 2, since all neighbors of vertex v is in A, then carrying v to the other side will decrease the cut-rank function value by one.

Thus, for every separation $(A, V \setminus A)$ of V s.t. $\{u\} \subset A$ and $\{v\} \subset (V \setminus A)$, $\rho(A, V \setminus A) - 1 \leq \rho(A \cup \{v\}, V \setminus (A \cup \{v\})) \leq \rho(A, V \setminus A)$. □

Theorem 4. *Let G be a graph and u and v be two neighbor vertices where $|N(u)| = |N(v)| = 2$, then there exists an optimal rank decomposition of G such that the leaves corresponding to u and v are connected to the same non-leaf node.*

Proof. Since Theorem 3 holds for every separation $(A, V \setminus A)$ of V s.t. $\{u\} \subset A$ and $\{v\} \subset (V \setminus A)$ then there is no additional cost for forcing vertices u and v to be in the same separation (for the separations, where the sizes of both sides are at least 2) on the rank decomposition tree. To have u and v connected to the same non-leaf node on the rank decomposition does not increase the width of the rank decomposition. Hence, Theorem 4 follows. □

Using Theorem 4, in the reduction procedure we can contract the edge between two vertices of degree two if they are neighbors, and connect them to the same non-leaf node in the rank decomposition. The obtained separation will have a cut-rank function value of 2.

VI.4.3. Reduction procedure

The complete reduction procedure is as follows. At each step we first search for a degree one vertex, and if there is one we contract the incident edge to that vertex, and connect the subtrees corresponding to the degree one vertex and its neighbor to the same new non-leaf node. If there is no degree one vertex then we search for twins. If there is a twin pair, then we identify those two vertices and connect the subtrees corresponding to the twins to the same new non-leaf node. If there are no twins either, then we search for two neighbor vertices of degree two, identify these two vertices and connect the subtrees corresponding to these vertices to the same new non-leaf node. If a subtree we deal with in this process consists of a single vertex then we can connect it to the appropriate non-leaf node by an edge. If it is larger than a single vertex, then we connect the degree two non-leaf node in this subtree to the appropriate non-leaf node by an edge. The reduction procedure stops when there is no degree one vertex, or twins, or two neighbor vertices of degree two.

VI.5. Computational results

We use the same test instances we use for the branchwidth problem. Additionally, we use “tele” graphs provided by W. J. Cook and some test instances we have generated. We run the rank-width heuristic using C++ on a Pentium 4 computer with 1.60 GHz CPU. We run the model on rank-width problem using C++ and CPLEX 9.0 on a Pentium 4 computer with 3.06 GHz CPU.

Table 19 and 20 show the performance of the rank-width heuristic and compare the bounds with the bounds obtained by the hybrid method. The columns “hm”,

Graph	$(\|V\|, \|E\|)$	hm	timehm	rwh	timerwh
dens	(6,7)	2	0	2	0
$W_{6,2}$	(6,9)	3	0	2	0
K_5	(5,10)	4	0	1	0
cosqb	(8,10)	2	0.1	2	0.1
$W_{8,3}$	(8,12)	4	0	3	0.1
coeray	(11,14)	2	0	2	0.2
sudtbl	(11,14)	3	0	3	0.2
K_6	(6,15)	4	0	1	0.1
$W_{10,4}$	(10,15)	4	0.1	4	0.2
Petersen	(10,15)	4	0.1	4	0.2
arret	(12,15)	2	0	2	0.3
supp	(12,15)	2	0	2	0.3
$W_{12,5}$	(12,18)	4	0.1	4	0.3
laspow	(14,18)	3	0.1	3	0.5
tpart	(14,18)	2	0	2	0.8
inter	(16,21)	2	0	2	0.8
nprio	(17,22)	3	0	3	1
saturr	(17,23)	2	0.1	2	1.5
setinj	(18,23)	2	0	3	1.4
genb	(18,24)	2	0	2	1.3
laser	(19,25)	2	0	2	1.6
drigl	(21,29)	3	0.1	3	1.9
si	(23,31)	3	0	3	3.7
seval	(24,31)	3	0	3	4.9
bcndb	(25,34)	2	0	2	4.9
dyeh	(27,35)	3	0.1	3	7.7
linj	(27,35)	3	0	3	6.2
recre	(27,35)	3	0	3	7
injchk	(33,44)	2	0.1	2	24.8
bilan	(37,49)	3	0.1	3	26.6
tcomp	(38,50)	3	0.1	3	27.6
colbur	(40,54)	3	0	4	32.3
sortie	(44,62)	3	0	4	51.1
cardeb	(48,64)	3	0.1	3	71.5
prophy	(48,65)	2	0.1	4	67.9
yeh	(50,68)	3	0.1	3	80.8
verify	(55,73)	2	0.1	3	117.3
bcnd	(55,77)	2	0	3	120.6
heat	(69,95)	3	0.1	3	389.6
ddeflu	(92,124)	2	0.2	4	941.3
pastem	(158,214)	3	0.4	5	8131.4

Table 19 : Rank-width heuristic run times and bounds

Graph	$(\|V\|, \|E\|)$	hm	timehm	rwh	timerwh
g11_31	(11,31)	5	0	4	0.4
g14_52	(14,52)	7	0	5	1
g19_33	(19,33)	4	0	4	2.6
g19_88	(19,88)	10	0	7	3.5
tele20	(20,21)	2	0	1	2.6
g20_119	(20,119)	12	0.1	8	4.6
g25_92	(25,92)	11	0.1	9	9.5
g25_168	(25,168)	15	0.2	9	11
g28_63	(28,63)	6	0	6	12.7
g34_221	(34,221)	16	0.4	12	36.4
tele39	(39,90)	7	0.1	6	48.5
g40_239	(40,239)	19	0.8	14	65.3
g43_284	(43,284)	20	0.7	15	91
g45_88	(45,88)	8	0.1	7	82.7
g45_325	(45,325)	23	0.8	15	112.797
g52_100	(52,100)	8	0.2	10	144.5
g53_86	(53,86)	9	0.1	11	152.6
tele53	(53,97)	6	0.2	6	156
g54_90	(54,90)	9	0.2	9	165
tele56	(56,85)	3	0.1	3	186.7
g57_696	(57,696)	33	1.6	19	331.2
g57_785	(57,785)	36	2	19	345.1
g58_577	(58,577)	31	2.2	20	338.1
g58_788	(58,788)	39	0.6	19	364.3
g59_729	(59,729)	36	1.3	20	387.1
g59_822	(59,822)	37	1.6	20	395.7
g60_112	(60,112)	10	0.4	13	255.7
tele62	(62,138)	6	0.2	8	299.4
g62_930	(62,930)	41	1.2	21	501
g63_986	(63,986)	41	1.1	21	529.4
g64_1021	(64,1021)	41	1.6	22	570.3
g67_950	(67,950)	41	2.1	23	653.4
g81_141	(81,141)	12	0.8	15	835
tele89	(89,144)	7	0.1	6	1180.1
tele133	(133,212)	5	0.7	8	6008.6
tele226	(226,288)	5	0.8	NA	X

Table 20 : Rank-width heuristic run times and bounds for additional test instances

"timehm", "rwh" and "timerwh" represent the bound obtained by the hybrid method, the running time of the hybrid method, the bound obtained by the rank-width heuristic and the running time of the rank-width heuristic, respectively. The run times are given in seconds. An "X" on any of the tables means that the model does not terminate before 18000 seconds.

We use a Pentium 4 computer with 1.60 GHz CPU to obtain our test instances. As we can see in Table 19 and 20, the run times of the rank-width heuristic increase considerably with the size of the test instance. The bounds obtained by the rank-width heuristic is smaller than the bounds obtained by the hybrid method for many of the test instances. For larger sparse graphs hybrid method performs better. The rank-width heuristic performs better on dense graphs. Branchwidth of a graph is a tight bound on the rank-width of that graph. The effectiveness of hybrid method and rank-width heuristic on different graphs might lead to a conclusion that the gap between the branchwidth and the rank-width of a graph increases as the graph gets denser. In terms of time requirement hybrid method outperforms the rank-width heuristic. But this is a trade off since we will be saving some model run time if we get a good lower bound by spending more time. In our model we use both methods and set the smaller bound we obtain from the two heuristics as an upper bound.

Table 21 shows the rank-width, building and running times for the test instances using the third model to compute rank-width. We use a Pentium 4 computer with 3.06 GHz CPU to obtain our results. The column $|V'|$ represents the number of vertices in the graph obtained after preprocessing steps. The columns "rw", "buildno", "buildw", "runno", "runw" represent the rank-width, building time with no upper bound heuristic and reductions, building time with upper bound heuristic and reductions, running time with no upper bound heuristic and reductions and running time with upper bound heuristic and reductions, respectively.

Graph	$(\|V\|, \|E\|)$	$\|V'\|$	rw	buildno	buildw	runno	runw
dens	(6,7)	3	2	0.5	0	0.1	0
$W_{6,2}$	(6,9)	6	2	0.5	0	0.1	0
K_5	(5,10)	NA	1	0.5	0	0.1	0
cosqb	(8,10)	3	2	0.8	0.1	0.2	0
$W_{8,3}$	(8,12)	8	3	1	0.8	1.5	0.4
coeray	(11,14)	8	2	4.3	0	0.2	0
sudtbl	(11,14)	10	3	6.5	1	3432.6	0.8
K_6	(6,15)	NA	1	0.5	0	0.1	0
$W_{10,4}$	(10,15)	10	3	6.6	1.9	50.2	18.2
Petersen	(10,15)	10	3	6.9	1.8	67.7	16.5
arret	(12,15)	3	2	9	0.1	0.3	0
supp	(12,15)	3	2	8.8	0.1	0.4	0
$W_{12,5}$	(12,18)	12	3	36.5	5.5	17158.8	597.8
laspow	(14,18)	10	2	54.4	1.8	X	0.7
tpart	(14,18)	3	2	38	0.2	1035.2	0
inter	(16,21)	13	2	165.8	0.1	2731.9	0
nprio	(17,22)	10	2	458.2	1.6	X	0.1
saturr	(17,23)	10	2	363.8	0.2	0.9	0
setinj	(18,23)	8	2	767.1	1.1	X	0.2
genb	(18,24)	10	2	766.8	0.5	X	0
laser	(19,25)	12	2	1696.3	0.3	1.39	0
drigl	(21,29)	13	3	7931.1	9.7	18010.8	4
si	(23,31)	16	2	X	91	NA	0.3
seval	(24,31)	20	NA	X	1699.3	NA	X
bcndb	(25,34)	11	2	X	1.2	NA	0
dyeh	(27,35)	12	3	X	7.7	NA	13.1
linj	(27,35)	12	3	X	10.2	NA	50
recre	(27,35)	24	NA	X	X	NA	NA
injchk	(33,44)	17	2	X	2.8	NA	0
bilan	(37,49)	29	NA	X	X	NA	NA
tcomp	(38,50)	34	NA	X	X	NA	NA
colbur	(40,54)	27	NA	X	X	NA	NA
sortie	(44,62)	33	NA	X	X	NA	NA
cardeb	(48,64)	26	NA	X	X	NA	NA
prophy	(48,65)	31	2	X	X	NA	NA
yeh	(50,68)	28	NA	X	X	NA	NA
verify	(55,73)	48	2	X	X	NA	NA
bcnd	(55,77)	37	2	X	X	NA	NA
heat	(69,95)	37	NA	X	X	NA	NA
ddeflu	(92,124)	57	2	X	X	NA	NA
pastem	(158,214)	100	NA	X	X	NA	NA

Table 21 :Model 3 building and running times for the rank-width

Table 21 shows the effectiveness of the upper bound and the reductions. For all of the test instances, both building times and the running times are much smaller when upper bound and reductions are used. Building procedure includes the construction of separations with cut-rank function value less than or equal to the upper bound. Hence, the building times of the test instances with more than 26 vertices are greater than 18,000 seconds, if they have a rank-width heuristic bound greater than 2. For the test instance “injchk”, the model builds and runs quickly because the rank-width heuristic bound is 2. The run times can be very long even for a graph with 24 vertices, such as the test instance “seval”, depending on the number of reduction steps implemented. The reduction procedure can reduce “seval” to a graph of 20 vertices. Obtaining all separations for this graph takes 1699.3 seconds. However for a larger graph such as “linj” with 27 vertices, it takes only 10.2 seconds to build the model because the reduced graph is smaller than the reduced graph of “seval”.

CHAPTER VII

CONCLUSION AND FUTURE WORK

We implemented three integer programming models for the branchwidth problem. The first two models are very similar although the second model has fewer constraints in the main formulation. In addition, the second model requires some constraints to be added as cuts when these constraints are violated. There are $O(m^3)$ constraints in the first model, and $O(nm^2)$ constraints in the main formulation of the second model. Also, Model 2 has fewer variables. In general, Model 2 performs better than Model 1 in terms of run time. The building times can be very close, but the run times of the second model are shorter than the run times of the first model in many of the test instances.

We run our models using C++ and CPLEX 9.0 on a Pentium 4 computer with 3.06 GHz CPU. We set a time limit of 5 hours for the termination of all three models. If the graph test instance can be reduced to a very small graph, like in the case of compiler graphs, both models perform well. In contrast, they may not perform very well when the graph has more than 11 edges and a minimum degree larger than two. For those graphs that can be reduced to smaller graphs, depending on the structure of the new graph, the second model can run to the completion in less than 18,000 seconds, which is equivalent to 5 hours. For the same graphs, Model 1 can take more than 5 hours to terminate. One possible reason is that the second model has fewer variables and fewer constraints and it is easier to verify feasibility with fewer constraints. Also, the number of branches that the model needs to go through is smaller when we have fewer variables. However, for graphs such as M_8 and Q_3 both models run for more than 5 hours although these graphs have only 12 edges.

The third model can perform well for graphs with up to 23 edges. The building times are longer compared to the first two models, though still less than 5 hours. The building times grow exponentially with the number of edges, and become more than 5 hours when we have more than 23 edges. This model has fewer variables, and fewer constraints than the first two formulations for small graphs. The drawback is the time required for obtaining all connected separations. When we compare this model with the first two models, depending on the structure of the graph after reductions Model 2 and Model 3 usually result in short running times. However, the model building times are much longer in Model 3. In our computations, we observed that for the graphs with branchwidth 4, the third model runs much faster. However, for the graphs with branchwidth 5, the second model is better.

We have applied upper bound and lower bound heuristics to all three models. We obtained upper bounds equal to lower bounds for all of the compiler graphs. For many other test instances this is the case. For these graphs all three models use the branch decomposition obtained by the upper bound heuristic as the optimal branch decomposition. Thus, the run times were less than a second. For the remainder of the test instances, both heuristics still perform very well, and can attain a gap of 1 for most of the test instances. However, even for a gap of 1, the first two models can run more than 5 hours for graphs as small as M_8. Branchwidth of maximum planar subgraph of the input graph is another lower bound that could be considered for our models. However, we did not implement any further methods to compute lower bounds since when the lower bound is not equal to the upper bound, for slightly larger graphs we will not be able to reach the optimal solution in less than 5 hours using any of the three models.

Moreover, we used some reduction techniques to reduce the size of the input graph, and the size of the model as well. These reductions are based on the degree one and degree two vertices. We implemented some cuts for the first two models, and compared their performances on some test instances. The experiments indicate that in general local and δ cuts together perform the best compared to the other cuts, or the no cuts option.

A difficulty we encountered was generating test instances that would have a gap of at least 1 between their upper and lower bounds, and run less than 5 hours. But, at the same time these test instances need to run sufficiently long such that the effect of the time spent for generating cuts is balanced, so that we can have a realistic comparison of the different cuts. Branchwidth problem on small graphs, or graphs that can be reducible to small graphs can be solved in less than a second. The upper bound is equal to the lower bound for these graphs. For graphs as small as M_8 or the Petersen graph, the first two models run for more than 5 hours. However, for larger graphs such as $g11\_23$ and $g19\_33$ the second model solves the problem in less than 5 hours. For even larger graphs all three models run for longer than 5 hours. Thus, generating test instances to have a good comparison of the three models was challenging as well. This is the reason behind having a limited number of test instances for comparison purposes.

Furthermore, Hicks [50] implemented a branch decomposition-based algorithm to compute the branchwidth and an optimal branch decomposition of a graph. In his paper the run times for some test instances with up to 919 edges are illustrated. For the test instances with up to 45 edges his algorithm solves the instance in less than 20 seconds. For larger graphs the algorithm can still achieve the optimal solution in less than 1000 seconds for most of the illustrated test instances. Thus, Model 3 is not comparable to this algorithm since it cannot build the test instances with more than 23 edges in less than 5 hours. Models 1&2 do not perform very well when the graph cannot be reduced to a smaller graph. Thus, compared to the three integer programming models we have Hicks' branch decomposition based algorithm is faster and can deal with much larger graphs, i.e., graphs with up to several hundred edges, depending on the input branch decomposition, which is found using a heuristic.

In addition to the implementation on graphs, we used the three models to compute the branchwidth and optimal branch decomposition of hypergraphs. Hicks's above mentioned algorithm is developed for graphs only. The third model performs the best in our experiments for hypergraphs. This is mainly because this model has fewer variables and fewer constraints compared to the other two models. Our comparison was based on the hypergraphs with up to 13 hyperedges. Thus, the building times for the third model were reasonable as well. A direction for future research on all three models is finding appropriate reduction techniques for hypergraphs, and targeting larger hypergraphs.

For the branchwidth problem in general, all the models are open to be improved to obtain reasonable running times for larger graphs or hypergraphs. However, we need to mention that for a graph with m edges, there are $(2m-5)!!$ possible branch decompositions ("n!!" is equal to the multiplication of the odd numbers up to n). For a graph with 10 edges, there exists more than 2 million possible branch decompositions. This indicates the degree of difficulty of the branchwidth problem.

Other than the work done on branchwidth problem, we implemented the third model for the rank-width problem. We applied some reduction procedures on degree one and degree two vertices as well as twins. The model performs very well for compiler graphs with up to 24 nodes. For larger graphs, depending on the size of the graph after the reduction procedure and the bound obtained by the upper bound heuristics, it can handle compiler graphs with up to 33 nodes. Before we applied the reductions and the upper bound heuristics, the model could handle compiler graphs with up to 14 vertices, or sometimes up to 21 vertices, depending on the structure of the test instance.

Moreover, we implemented a rank-width heuristic to increase the efficiency of our model. This heuristic outputs a feasible integer solution for the rank-width problem. This output serves as a possible upper bound for our model. Branchwidth of a graph is another upper bound for the rank-width of that graph. Thus, we use the best bound obtained by a branchwidth heuristic (hybrid method in this case) and the rank-width heuristic. The rank-width heuristic runs longer than the hybrid method, but the model can benefit greatly from the solution, especially if the output of the rank-width heuristic is smaller than or equal to the output of the hybrid method, because the rank-width heuristic also provides a rank decomposition, but hybrid method does not.

Run times of the test instances using the third model on the rank-width problem can be reduced by adding a good lower bound and more reduction techniques to the model. Eliminating a set of separations, like in the case of connected separations for the branchwidth problem, would narrow the feasible region, and reduce the running times even more. Building times may increase, and this can be a trade off. Reducing the running time of the third model on the rank-width problem by investigating the possible reductions, cuts, and lower bounds can be a direction for future research. Building another model for the rank-width problem can be another direction. The model we have implemented for the rank-width problem can be a basis of comparison for the future models.

REFERENCES

1. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. *Journal of Algorithms* **12**, 308–340 (1991)

2. Baltz, A., Srivastav, A.: Fast approximation of minimum multicast congestion - implementation versus theory. *RAIRO Operations Research* **38**, 319–344 (2004)

3. Behjat, L., Kucar, D., Vannelli, A.: A novel eigenvector technique for large scale combinatorial problems in vlsi layout. *Journal of Combinatorial Optimization* **6**(3), 271–286 (2004)

4. Behjat, L., Vannelli, A., Rosehart, W.: Integer linear programming models for global routing. *INFORMS Journal on Computing* **18**(2), 137–150 (2006)

5. Bian, Z., Gu, Q., Marzban, M., Tamaki, H., Yoshitake, Y.: Empirical study on branchwidth and branch decomposition of planar graphs. *Proc. of the 2008 SIAM Workshop on Algorithm Engineering and Experiments (ALENEX08)* (2008). To appear

6. Bodlaender, H.: A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal of Computing* **25**, 1305–1317 (1996)

7. Bodlaender, H., Kloks, T.: Efficient and constructive algorithms for the path-width and treewidth of graphs. *Journal of Algorithms* **21**, 358–402 (1996)

8. Bodlaender, H.L., Koster, A.M.C.A., Wolle, T.: Contraction and treewidth lower bounds. *Technical Report 34*, UU-CS, Dept. of Computer Science, Utrecht University, Utrecht, The Netherlands (2004)

9. Bodlaender, H.L., Thilikos, D.M.: Constructive linear time algorithms for branch-width, *Lecture Notes in Computer Science*, vol. 1256, pp. 627–637. Springer-Verlag, Berlin, Germany (1997)

10. Bodlaender, H.L., Thilikos, D.M.: Graphs with branchwidth at most three. *Journal of Algorithms* **32**, 167–194 (1999)

11. Bouchet, A.: Circle graph obstructions. *Journal of Combinatorial Theory, Series B* **60(1)**, 107–144 (1994)

12. Carr, R., Vempala, S.: Randomized metarounding. *Proc. of the 32nd ACM Symposium on the Theory of Computing (STOC'00)*, pp. 58–62. Portland, USA (2000)

13. Cheriyan, J., Salavatipour, M.R.: Hardness and approximation results for packing Steiner trees, *Lecture Notes in Computer Science*, vol. 3221, pp. 180–191. Springer, Berlin-Heidelberg (2004)

14. Cheriyan, J., Salavatipour, M.R.: Packing element-disjoint Steiner trees, *Lecture Notes in Computer Science*, vol. 3624, pp. 52–61. Springer, Berlin-Heidelberg (2005)

15. Chopra, S.: Comparison of formulations and a heuristic for packing Steiner trees in a graph. *Annals of Operations Research* **50**, 143–171 (1994)

16. Cook, W.J., Seymour, P.D.: An algortihm for the ring-router problem. *Technical Report*, Bellcore, Morristown, NJ (1994)

17. Cook, W.J., Seymour, P.D.: Tour merging via branch-decomposition. *INFORMS Journal on Computing* **15(3)**, 233–248 (2003)

18. Cornuéjols, G.: Combinatorial optimization packing and covering. *Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, Carnegie Mellon University, Pittsburgh, Pennsylvania (2001)

19. Courcelle, B.: The monadic second-order logic of graphs I: Recognizable set of finite graphs. *Information and Computation* **85**, 12–75 (1990)

20. Courcelle, B., Olariu, S.: Upper bounds to the clique width of graphs. *Discrete Appl. Math.* **101**(1-3), 77114 (2000)

21. Courcelle, B., Oum, S.: Vertex-minors, monadic second-order logic, and a conjecture by Seese. *Journal of Combinatorial Theory, Series B* **97**(1), 91126 (2007)

22. Cunningham, W.: Minimum cuts, modular functions, and matroid polyhedra. *Networks* **15**(2), 205–215 (1985)

23. Cunningham, W.H., Geelen, J.: On integer programming and the branch-width of the constraint matrix, *Lecture Notes in Computer Science*, vol. 4513, pp. 158–166. Springer, Berlin, Heidelberg (2007)

24. Demaine, E.D., Fomin, F.V., Hajiaghayi, M.T., Nishimura, N., Rgade, P., Thilikos, D.M.: Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. *Journal of Computer and System Sciences* **69**, 166–195 (2004)

25. Demaine, E.D., Fomin, F.V., Hajiaghayi, M.T., Thilikos, D.M.: Fixed-parameter algorithms for the (k, r)-center in planar graphs and map graphs. *ACM Transactions on Algorithms* **1**, 33–47 (2005)

26. Dorn, F.: Special branch decompositions: A natural link between tree decompositions and branch decompositions. Ph.D. thesis, University of Bergen, Norwegen (2004)

27. Dorn, F.: Dynamic programming and fast matrix multiplication, *Lecture Notes in Computer Science*, vol. 4059, pp. 280–291. Springer, Berlin, Heidelberg (2006)

28. Dorn, F., Fomin, F.V., Thilikos, D.M.: Fast subexponential algorithm for non-local problems on graphs of bounded genus, *Lecture Notes in Computer Science*, vol. 4168, pp. 172–183. Springer, Berlin, Heidelberg (2006)

29. Dorn, F., Telle, J.A.: Two birds with one stone: the best of branchwidth and treewidth with one algorithm, *Lecture Notes in Computer Science*, vol. 3887, pp. 386–397. Springer, Berlin, Heidelberg (2005)

30. Fellows, M.R., Rosamond, F.A., Rotics, U., Szeider, S.: Clique-width minimization is NP-hard. In: *38th annual ACM Symposium on Theory of Computing*, pp. 354–362. ACM Press, New York, NY, USA (2006)

31. Fomin, F., Mazoit, F., Todinca, I.: Computing branchwidth via efficient triangulations and blocks, *Lecture Notes in Computer Science*, vol. 3787, pp. 374–384. Springer, Berlin, Heidelberg (2005)

32. Fomin, F.V., Fraigniaud, P., Thilikos, D.M.: The price of connectedness in expansions. *Technical Report LSI-04-28-R*, Departament de Lenguatges i Sistemes Informàtics, Universitat Politécnica de Catalunya, Spain (2004)

33. Fomin, F.V., Thilikos, D.M.: Fast parameterized algorithms for graphs on surfaces: Linear kernel and exponential speed-up, *Lecture Notes in Computer Science*, vol. 3142, pp. 581–592. Springer, Berlin, Heidelberg (2004)

34. Fomin, F.V., Thilikos, D.M.: A simple and fast approach for solving problems on planar graphs, *Lecture Notes in Computer Science*, vol. 2996, pp. 56–67. Springer, Berlin, Heidelberg (2004)

35. Garey, M., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco (1979)

36. Garey, M.R., Graham, R.L., Johnson, D.S., Knuth, D.E.: Complexity results for bandwidth minimization. *SIAM Journal of Applied Mathematics* **34**, 477–495 (1978)

37. Geelen, J., Gerrards, A., Robertson, N., Whittle, G.: On the excluded minors for the matroids of branch-width k. *Journal of Combinatorial Theory, Series B* **88**, 261–265 (2003)

38. Geelen, J., Gerrards, A., Robertson, N., Whittle, G.: Obstructions to branch-decomposition of matroids. *Journal of Combinatorial Theory, Series B* **96(4)**, 560–570 (2006)

39. Geelen, J., Gerrards, A., Whittle, G.: Branch width and well-quasi-ordering in matroids and graphs. *Journal of Combinatorial Theory, Series B* **84**, 270–290 (2002)

40. Grötschel, M., Martin, A., Weismantel, R.: Packing Steiner trees: A cuting plane algorithm and computational results. *Mathematical Programming* **72**, 125–145 (1996)

41. Grötschel, M., Martin, A., Weismantel, R.: Packing Steiner trees: Further facets. *European Journal of Combinatorics* **17(1)**, 39–52 (1996)

42. Grötschel, M., Martin, A., Weismantel, R.: Packing Steiner trees: Polyhedral investigations. *Mathematical Programming* **72**, 101–123 (1996)

43. Grötschel, M., Martin, A., Weismantel, R.: Packing Steiner trees: Separation algorithms. *SIAM Journal on Discrete Mathematics* **9(2)**, 233–257 (1996)

44. Grötschel, M., Martin, A., Weismantel, R.: The Steiner tree packing problem in VLSI design. *Mathematical Programming* **78**, 265–281 (1997)

45. Gu, Q., Tamaki, H.: Optimal branch-decomposition of planar graphs in O(n3) time, *Lecture Notes in Computer Science*, vol. 3580, pp. 373–384. Springer, Berlin, Heidelberg (2005)

46. Hall, R., Oxley, J., Semple, C., Whittle, G.: On matroids of branch-width three. *Journal of Combinatorial Theory, Series B* **86**, 148–171 (2002)

47. Hicks, I.V.: Branch decompositions and their applications. Ph.D. thesis, Rice University, Houston, TX (2000)

48. Hicks, I.V.: Branchwidth heuristics. *Congressus Numerantium* **159**, 31–50 (2002)

49. Hicks, I.V.: Branch decompositions and minor containment. *Networks* **43(1)**, 1–9 (2004)

50. Hicks, I.V.: Graphs, branchwidth, and tangles! Oh my! *Networks* **45(2)**, 55–60 (2005)

51. Hicks, I.V.: Planar branch decompositions I: The ratcatcher. *INFORMS Journal on Computing* **17(4)**, 402–412 (2005)

52. Hicks, I.V.: Planar branch decompositions II: The cycle method. *INFORMS Journal on Computing* **17(4)** (2005)

53. Hicks, I.V., Jr., N.B.M.: The branchwidth of graphs and their cycle matroids. *Journal of Combinatorial Theory, Series B* **97**(5), 681–692 (2007)

54. Hliněný, P.: On the excluded minors of matroids of branch-width three. *Electr. Journal of Combinatorics* **9** (2002)

55. Hliněný, P.: A parameterized algorithm for matroid branch-width. *SIAM Journal of Computing* **35(2)**, 259–277 (2005)

56. Hliněný, P., Oum, S.: Finding branch decompositions and rank decompositions, *Lecture Notes in Computer Science*, vol. 4698, pp. 163–174. Springer, Berlin, Heidelberg (2007)

57. Hliněný, P., Seese, S.O.D., Gottlob, G.: Width parameters beyond tree-width and their applications. *Computer Journal* **2007**(online), 1–37 (2007)

58. Jain, K., Mahdian, M., Salavatipour, M.R.: Packing steiner trees. In: *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 266–274. Baltimore, Maryland (2003)

59. Jansen, K., Zhang, H.: Approximation algorithms for general packing problems and their application to the multicast congestion problem. *Mathematical Programming* **114**(1), 183–206 (2008)

60. Kloks, T., Kratochvil, J., Mller, H.: Computing the branchwidth of interval graphs. *Discrete Applied Mathematics* **145**, 266–275 (2005)

61. Koch, T., Martin, A.: Solving Steiner tree problems in graphs to optimality. *Networks* **32**(3), 207–232 (1998)

62. Lagergren, J.: Efficient parallel algorithms for tree-decomposition and related problems. In: *Proc. of 31st Ann. Sympos. on the Foundations of Comput. Sci.*, vol. 1, pp. 173–182. ACM Press, New York (1990)

63. Lengauer, T., Lügering, M.: Integer program formulations of global routing and placement problems. *Research Report*, University of Paderborn, Germany (1991)

64. Lengauer, T., Lügering, M.: Provably good global routing of integrated circuits. *SIAM Journal of Optimization* **11(1)**, 1–30 (2000)

65. Matoušek, J., Thomas, R.: Algorithms finding tree-decompositions of graphs. *Journal of Algorithms* **12**, 1–22 (1991)

66. Mazoit, F.: The branch-width of circular-arc graphs, *Lecture Notes in Computer Science*, vol. 3887. Springer, Berlin, Heidelberg (2006)

67. Mazoit, F., Thomassě, S.: Branchwidth of graphic matroids. *London Mathematical Society Lecture Note Series.* 346. Cambridge University Press, Cambridge (2007)

68. Monien, B.: The bandwidth minimization problem for caterpillars with hair length 3 is NP-Complete. *SIAM J. Alg. Disc. Meth.* **7**, 505–512 (1986)

69. Oum, S.: Rank-width and vertex-minors. *Journal of Combinatorial Theory, Series B* **95**(1) (2005)

70. Oum, S.: Approximating rank-width and clique-width quickly. *J. Combinatorial Theory Series B* **96**(4) (2006)

71. Oum, S.: Rank-width is less than or equal to the branchwidth. *J. of Graph Theory* **57**(3) (2008)

72. Oum, S., Seymour, P.: Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B* **96(4)**, 514–528 (2006)

73. Oum, S., Seymour, P.: Testing branch-width. *Journal of Combinatorial Theory, Series B* **97(3)**, 385–393 (2007)

74. Padberg, M., Rinaldi, G.: An efficient algorithm for the minimum capacity cut problem. *Mathematical Programming* **47**, 19–36 (1990)

75. Paul, C., Proskurowski, A., Telle, J.A.: Generation of graphs with bounded branchwidth, *Lecture Notes in Computer Science*, vol. 4271, pp. 205–216. Springer, Berlin, Heidelberg (2006)

76. Paul, C., Telle, J.A.: New tools and simpler algorithms for branchwidth, *Lecture Notes in Computer Science*, vol. 3669, pp. 379–390. Springer, Berlin, Heidelberg (2005)

77. Reed, B.: Finding approximate separators and computing treewidth quickly. In: *Proc. 24th Annual ACM Sympos. on the Theory of Comput.*, pp. 221–228. ACM Press, New York (1992)

78. Robertson, N., Seymour, P.: Graph minors IV: Treewidth and well-quasi-ordering. *Journal of Combinatorial Theory, Series B* **48**, 227254 (1990)

79. Robertson, N., Seymour, P.: Graph minors. X. obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B* **52**, 153–190 (1991)

80. Robertson, N., Seymour, P.D.: Graph minors XIII: The disjoint paths problem. *Journal of Combinatorial Theory, Series B* **63**, 65–110 (1995)

81. Seymour, P.D., Thomas, R.: Call routing and the ratcatcher. *Combinatorica* **14(2)**, 217–241 (1994)

82. Tamaki, H.: A linear time heuristic for the branch decomposition of planar graphs. *Technical Report*, MPI-I-2003-1-010, Max-Planck-Institut Fur Informatick, Saarbrücken, Germany (2003)

83. Terlaky, T., Vanelli, A., Zhang, H.: On routing in VLSI design and communication networks, *Lecture Notes in Computer Science*, vol. 3827, pp. 1051–1060. Springer, Berlin, Heidelberg (2005)

84. Vempala, S., Vöcking, B.: Approximationg multicast congestion, *Lecture Notes in Computer Science*, vol. 1741, p. 367. Springer, Berlin, Heidelberg (1999)

85. Wang, C., Liang, C., Jan, R.: Heuristic algorithms for packing of multiple-group multicasting. *Computers & Operations Research* **29**, 905–924 (2000)

VITA

Elif Ulusal was born to Abdulkadir Kolotoglu and Saime Kolotoglu in Trabzon, Turkey on July 18th, 1980. She graduated from H.S. in Trabzon in 1998. She received a B.S. in industrial engineering from Bilkent University, Ankara, Turkey in 2002. In August 2002, she started the Ph.D. program at the Industrial and Systems Engineering Department of Texas A&M University in College Station. She received the Ph.D. in May 2008. During her doctoral studies, she held positions as research assistant, teaching assistant and lecturer. She worked on research in collaboration with professors Dr. Illya Hicks, and Dr. Sergiy Butenko. Her research interests are in integer programming, combinatorial optimization, graph theory, and operations research.

Permanent address:

Besirli 2 Mah.
Esref Bitlis Cad.
Cinar Sit. C Blok No:13
Trabzon, TURKEY

This document was typeset in LaTeX by the author.