THE SLIM BRANCH AND PRICE METHOD WITH AN APPLICATION TO THE HAMILTONIAN P-MEDIAN PROBLEM

A Dissertation

by

AHMED MOHAMED BADR ALY MARZOUK

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

DOCTOR OF PHILOSOPHY

Chair of Committee,	Halit Üster
Co-Chair of Committee,	Erick Moreno-Centeno
Committee Members,	Sila Çetinkaya
	Nancy Amato
Head of Department,	Mark Lawley

December 2016

Major Subject: Industrial Engineering

ABSTRACT

The main objective of this dissertation is to present a new exact optimization method, the Slim Branch and Price (SBP) method, which is an improvement over the traditional Branch and Price (B&P) framework. SBP can be used to solve a large class of combinatorial optimization problems that can be solved by B&P type algorithms and that have binary master problems with fixed support (i.e., the sum of the variables in any feasible solution is fixed). This is an important class of problems as it includes several classical and fundamental problems. Towards this objective, this dissertation develops three algorithms to solve an interesting optimization problem, the Hamiltonian p-median problem (HpMP), which is a generalization of the well-known traveling salesman problem. In HpMP, the target is to find p cycles that partition a given **undirected** graph with the objective of minimizing the total sum of the costs of these p cycles.

This dissertation is divided into three main parts with the objective of showing the superiority of SBP over B&P while using HpMP as a running example. Towards this objective, the first part presents a B&P algorithm for HpMP, the second part presents SBP and how it can be tailored to solve HpMP, and finally, the third part explains how the preprocessing techniques developed for integer programs can dramatically enhance the performance of SBP.

In the first part, we devise a Branch and Price algorithm that is able to solve instances with up to 318 nodes (within acceptable optimality gaps). To achieve this, we modified the set partitioning formulation of HpMP—a minor modification yet with significant algorithmic and computational advantages. Furthermore our computational results demonstrate that the practical complexity of HpMP and the

performance of the algorithms to solve it strongly depend on the value of p. In addition, in order to solve the pricing problem we make contributions on a couple of problems that are important on their own right: 1) we develop a new efficient algorithm to find the least cost cycle in undirected graphs with arbitrary edge costs and no negative cycles; and 2) we develop an algorithm to find the most negative cycle in undirected graphs with arbitrary edge costs. Finally, we prove that for every value of p, HpMP is NP-hard even when restricted to Euclidean graphs.

In the second part, we present SBP method which is an improvement over traditional B&P in the case of binary master problems with fixed support. The main advantage in SBP is that the branching tree has only one main branch with several leaves. In addition, we show that all the problems defined on the leaves can be merged to form a larger problem that can be solved very fast without further branching. We illustrate the computational advantage of SBP over B&P on HpMP. In particular, within one hour time limit, SBP can solve to optimality instances with up to 200 nodes; whereas B&P can solve to optimality instances with up to 127 nodes.

In the third part, we exploit the reduced cost fixing preprocessing technique to enhance the performance of B&P. To this end, we develop a heuristic based on k-opt moves to find good feasible solutions for HpMP. We also introduce two separation algorithms to improve the linear programming relaxation of the natural variable space model of HpMP. Using these upper and lower bounds, reduced cost fixing was then implemented to reduce the graph size by deleting the edges that cannot be in any optimal solution. We compared the computational times reported by SBP with preprocessing versus those reported by SBP without preprocessing. The former algorithm performed better than the latter algorithm in 88.3% of the test instances.

Dedicated to my mother, my father, and my sister for their endless support, continuous encouragement, and unconditional love.

ACKNOWLEDGEMENTS

I would like to thank my advisors, Dr. Halit Üster and Dr. Erick Moreno-Centeno, for all their support and patience during my PhD journey. They greatly helped me to be a better researcher. We spent hundreds of hours reciprocating and polishing ideas. I am indebted to them for their meticulous mentorship and continuous encouragement throughout the different phases of my doctoral degree. Moreover, I would also like to thank them for their major contributions in polishing my writing style to be simpler and more direct. Finally, I would like to thank them for their generosity in funding me throughout my five years at Texas A&M University.

I would like to thank Dr. Sila Çetinkaya and Dr. Nancy Amato for serving on my committee, their continuous encouragement throughout my studies, and their constructive comments for improving this dissertation. Besides, their lectures were really inspirational and exceptional. Moreover, I really appreciate their flexibility in setting my proposal and defense dates in spite of their very busy schedules.

I would like to thank all the members in the Industrial and Systems Engineering department at Texas A&M University. My special thanks to Dr. Guy Curry, Dr. Wilbert Wilhelm, and Dr. Kiavash Kianfar for their excellent lectures which taught me a lot. I would like to extend my thanks to Ms. Judy Meeks, Ms. Erin Roady, and Mr. Mark Hopcus for all their efforts in easing the obstacles faced by the graduate students.

I would like to thank all my friends in College Station who were really like family to me. The PhD journey would have been difficult without their presence and encouragement. Finally, I am really grateful to my mother, father, and sister whose unconditional love and support always helped me to pursue my PhD studies.

NOMENCLATURE

B&P	Branch and Price
LP	Linear Programming
RMP	Restricted Master Problem
HpMP	Hamiltonian p-median Problem
TSP	Traveling Salesman Problem
SBP	Slim Branch and Price
NVM	Natural Variable Space Model
E_t	Exploration Node t
R_t	Resolution Node t
OG	Optimality Gap
GNVM	Generalized Natural Variable Space Model
$\text{SBP}^{\text{p-2}}$	Slim Branch and Price with $L = p - 2$
$\text{SBP}^{\text{p-3}}$	Slim Branch and Price with $L = p - 3$
$\text{SBP}_{+}^{\text{p-2}}$	Slim Branch and Price with Reduced Cost Fixing and $L = p - 2$

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

1.1 Background and Motivation

In this dissertation, first, a combinatorial optimization algorithm, called Slim Branch and Price, is developed and its applications are discussed. Secondly, solution methods for an important optimization problem, called Hamiltonian p-median problem, are investigated. These methods include Branch and Price, Slim Branch and Price, and Slim Branch and Price with preprocessing.

1.1.1 Slim Branch and Price

The vast majority of the state-of-art successful exact algorithms for hard combinatorial optimization problems can be classified within either the branching framework or the cutting plane framework. On one hand, important examples of branching algorithms include Branch and Bound algorithms and Branch and Price (B&P) algorithms. On the other hand, pure cutting plane algorithms and Benders decomposition are important algorithms in the cutting plane framework. Some of the widely used effective methods combine ideas from these two frameworks as evident in Branch and Cut algorithms and Branch and Price and Cut.

B&P has been extensively used to solve several problems in many realms of operations research. Successful implementations of B&P in the areas of assignment, scheduling, vehicle routing, graph coloring, cutting stock, and multicommodity flow problems had been developed.

B&P was first presented by [8] to solve large scale optimization problems. B&P starts by formulating the respective problem as a problem with an exponential number of variables. This problem is called the master problem. This master problem is usually in the form of a set partitioning or covering problem with one or more

auxiliary constraints.

B&P starts by considering only a small subset of the set of all variables in the master problem to define a restricted master problem (RMP) from the master problem. Next, the linear programming (LP) relaxation of the RMP is solved and using its dual solution, a pricing problem is constructed. In minimization problem, the pricing problem entails finding a variable with negative reduced cost. If no such variable exists, then the optimal solution of the RMP's LP relaxation is also optimal for the master problem. Otherwise, a new variable (having a negative reduced cost) is added to the RMP and the RMP's LP relaxation is resolved. Branching starts when there are no variables with negative reduced cost and the solution of the RMP's LP relaxation is fractional. These same steps are applied (on possibly a slightly modified RMP and pricing problem) at each subsequent node in the branching tree until the optimal integer solution is found.

Many optimization problems were successfully solved using B&P and the literature discussing B&P algorithms is huge. Here, we outline some of the recent B&P successful implementations. Our literature review is not exhaustive, but our main aim is to show the wide applicability of B&P in diverse areas of applications.

Recent B&P algorithms were devised to solve the examination–timetabling problem [50], the maximum edge biclique packing problem on unbalanced bipartite graphs [1], maritime inventory routing [25], multi-trip vehicle routing problem with time windows and limited duration [24], genome rearrangement problems [29], the pickup and delivery problem with cross-docking [42], tramp ship routing and scheduling [48], weekly tour scheduling problem for postal service workers [11], kidney exchange problem with long chains [20], robust airline crew pairing problem [36], distance constrained multiple vehicle traveling purchaser problem [9], optimal allocation of emergency medical resources in a mass casualty incident [44], robust graph coloring

problem [5], the electricity production planning problem [39], two-echelon capacitated vehicle routing problem [41], the vehicle scheduling problem for fleets with alternative fuel vehicles [2], multi-skill project scheduling problem [35], among other problems.

The main objective of this dissertation is to present a new exact optimization method, the Slim Branch and Price (SBP) method, which is an improvement over the traditional Branch and Price framework. SBP can be used to solve a large class of combinatorial optimization problems that can be solved by B&P type algorithms and that have binary master problems with fixed support (i.e., the sum of the variables in any feasible solution is fixed). This is an important class of problems as it includes several classical and fundamental problems such as capacitated vehicle routing problem [7] and its variants, parallel machine scheduling [4] and its variants, capacitated p-median problem [32] and its variants, balanced disjoint rings problem [45], k-clustering problem [22], and political districting problem [34].

1.1.2 Hamiltonian p-median Problem

Another important optimization problem that has fixed binary support is the Hamiltonian p-median problem (HpMP) which is a generalization of the Traveling Salesman Problem (TSP) and was first presented by [10]. The target in HpMP is to find p cycles that partition a given **undirected** graph such that each node lies on exactly one cycle with the objective of minimizing the total sum of the costs of the p cycles. When the number of cycles p is not pre-specified, the problem is known as the two-matching problem and is solvable in polynomial time [18]. HpMP is equivalent to TSP when p is set to 1, and thus HpMP is, in general, NP-hard [21]. It was proved that for every value of p, HpMP is NP-complete [13] .

HpMP has many real life applications which are detailed as follows. In the hu-

manitarian logistics context, cycles represent the relief distribution routes through shelters or temporary evacuee locations and their connections along with travel times given as edge weights. Large relief shipments are typically dropped off at some node on each cycle and later distributed to the shelters via trucks. In our case, we form the routes (cycles) independent of any fixed drop-off locations set a priori to forming routes. Once the cycles are formed, any suitable node on a cycle is a potential drop-off location. This indeed provides great flexibility in changing conditions in emergency situations.

The HpMP also has application in scheduling jobs on parallel machines with sequence dependent set-up times. The relationship between the TSP and scheduling jobs with sequence dependent set-up times is well known [6]. Expanding on this relation, defining nodes as jobs and edges with setup times for switching job-to-job in an underlying network, HpMP seeks to optimally group nodes (jobs) into cycles and find a sequence of jobs for each group to be processed on a machine. Minimizing the total cycles-length corresponds to minimizing the total setup time required. Furthermore, [6] reviews TSP approaches for several flow-shop scheduling problems where each stage in the flow-shop includes a single machine. In [6], it is mentioned that an area in which no similar approaches are available is flexible flow-shop scheduling in which parallel processing at each stage is motivated by the availability of multiple machines. Thus, the HpMP is a step towards addressing this more general and relevant environment in today's manufacturing practice. A similar application may appear in the design of patrol routes and surveillance where a set of critical locations (facilities, intersections, hot crime spots etc.) needs to be visited periodically (e.g. [12], [38], [14]). Such patrol routes can be implemented for public-safety in cities and rural areas or overnight security services; in addition, these patrol routes are also relevant in military applications and border security.

Although HpMP has many applications as mentioned, only a limited number of (heuristic or exact) algorithms have been developed to solve it. The first effort to solve it heuristically was presented by [10]. They developed two formulations for HpMP and also several heuristics to solve it on directed graphs. Although they compared the output solution among the different heuristics, no results on the optimality gaps were reported.

Both [19] and [26] conducted a polyhedral study of the HpMP for their three-index model formulation and for the natural variable space formulation, respectively.

In [21], both theoretical and computational comparisons among seven new and existing formulations for HpMP were conducted. They successfully solved instances up to 40 nodes, but all the models had difficulty solving larger instances to optimality within the one hour time limit. Moreover, even the performance of the best model (the natural variable space formulation) deteriorated as the value of p increased even for instances with as few as 20 nodes. They also compared theoretically the relative tightness of the seven models and showed that both the set partitioning formulation and the natural variable space formulation have the tightest linear programming relaxations (however, the relative tightness of these two formulations was not comparable). They did not present any computational results for the set partitioning formulation. They pointed out that it requires column generation and left it as future research.

Similar problems to HpMP have been previously studied in the literature. In [30], a variant of the HpMP where p is an upper bound on the number of required cycles was presented. Another variant of HpMP was presented in [45], in which, in addition of specifying a predetermined number of cycles p, the formed cycles are required to be balanced in the sense that each cycle must pass through at least L and at most U nodes. Another variant was presented by [15] in which the number of cycles p is

not fixed but instead an upper bound on the number of nodes per cycle is imposed.

This dissertation is divided into three main parts. In the first part, we present a B&P algorithm to solve HpMP. In the second part, we present the SBP method and illustrate how it can be used to solve the HpMP. Finally, the third part presents the effect of implementing preprocessing techniques on the performance of SBP.

1.2 Contributions and Dissertation Organization

Next, we provide our motivation and contributions in each chapter of the dissertation separately.

In chapter 2, we develop and implement a B&P algorithm to solve HpMP. To this end, this chapter includes several contributions on modeling, methodology, and computational aspects:

1. We modified the set partitioning formulation of HpMP proposed by [21] without affecting the tightness of the LP relaxation. This simple modification greatly simplifies the pricing problem, thus effectively allowing us to solve larger instances;

2. We developed a new efficient algorithm to find the shortest cycle in an undirected graph with arbitrary edge costs and no negative cycles;

3. We developed an algorithm to find the most negative cycle in an undirected graph with arbitrary edge costs;

4. Computationally, the proposed algorithm for solving the HpMP outperformed the previously presented algorithms as it successfully solves instances up to 318 nodes, as opposed to other exact algorithms which only solved instances up to 100 nodes;

5. Since the proof in [13] only applies to general, incomplete graphs, we refined it to establish that for every value of p, HpMP is NP-hard even when restricted to Euclidean graphs;

6. We showed that the practical complexity of HpMP and the performance of the algorithms to solve it substantially depend on the relation between p and p_{2m} (the number of cycles in the minimum weight two-matching optimal solution).

Contributions 2 and 3 are relevant to our discussion because these two problems arise when solving the pricing problem. However, these two problems are important on their own right.

In chapter 3, we first present SBP method and its implementation in general and later we develop its specialization to solve HpMP. The computational results of SBP are compared with those obtained by the B&P algorithm presented in chapter 2. This comparison demonstrates the improved performance provided by the SBP in all of our test instances.

The new slim branching scheme is motivated by the following observations:

1. The traditional variable branching results in unbalanced branching tree in which the 0-branch often results in minimal improvement in the optimal objective value of the RMP's LP relaxation [46].

2. The solutions of the two children nodes in B&P branching tree are usually very close to the solution of their corresponding parent.

3. Most of the successful branching strategies implemented in B&P are based on exploiting the specific structure of the studied problem [46].

Observations 1 and 2 have the implication that most traditional branching strategies spend a lot of time examining a small portion of the feasible region. This results

in poor computational performance due to large branching tree. In SBP, we prevent these drawbacks as follows:

1. Although the branching strategy in SBP leads to an unbalanced tree, it guarantees that the problem defined at any left child node is easy and can be fathomed quickly without further branching.

2. The newly defined branching scheme guarantees that the optimal solution of the right child node differs significantly from the optimal solution of the parent's problem by the introduction of an inequality that sets an upper bound on the summation of the variables whose values are not zero in the parent's optimal solution. We refer to this inequality as the *exploration inequality*.

3. The branching scheme in SBP is not problem specific with the only requirement that the master problem has fixed binary support. SBP can be readily implemented to solve several applications provided that a pricing oracle that prevents the formation of specific columns (i.e., variables) is available.

In chapter 4, we use a well-known preprocessing technique, called the reduced cost fixing, for binary programs to further improve the performance of SBP when solving HpMP. To enhance the performance of reduced cost fixing technique, we suggest new approaches for improved lower and upper bounds. For the former, we present two new separation algorithms for a subset of constraints previously suggested for a natural variable model of HpMP [21]. For the latter, we present a heuristic based on k-opt moves to find a feasible solution for HpMP. Implementing our enhanced preprocessing technique enabled us to reduce the problem size by deleting a considerable number of edges in the original graphs, and thus allowed better solutions of equivalent, but significantly smaller size problems. Computational experiments

comparing the performance of SBP with and without preprocessing are presented to illustrate the significant improvements obtained.

In chapter 5, we present the concluding remarks and future research directions.

2. A BRANCH AND PRICE ALGORITHM FOR SOLVING THE HAMILTONIAN P-MEDIAN PROBLEM*

2.1 Introduction

The Hamiltonian p-median problem (HpMP) is a generalization of the Traveling Salesman Problem (TSP) and was first presented by [10]. The target in HpMP is to find p cycles that partition a given **undirected** graph such that each node lies on exactly one cycle with the objective of minimizing the total sum of the costs of the p cycles. When the number of cycles p is not pre-specified, the problem is known as the two-matching problem and is solvable in polynomial time [18]. HpMP is equivalent to TSP when p is set to 1, and thus HpMP is, in general, NP-hard [21]. We next show that for every value of p, HpMP is NP-hard for Euclidean graphs.

Proposition 2.1.1. *For every value of p, HpMP is NP-hard for Euclidean graphs.*

Proof. We perform a polynomial time reduction of TSP on Euclidean graphs to HpMP. Consider a Euclidean graph $G = (V, E)$ with edge lengths c_e for all edges $e \in E$ and let H be the sum of the $|V|$ highest edge lengths in G. Construct a new complete graph $G' = (V', E')$ by creating p copies of G and connecting the nodes in different copies with edges of length $(p+1)H$ to obtain G'. Clearly, the construction of G' from G can be done in polynomial time. Note that G' is also Euclidean.

Next, we show that the TSP instance on G has an optimal solution of value Z if and only if the HpMP instance on G', given a specific value of p, has an optimal solution of value pZ. Suppose that an optimal TSP solution on graph G has a value

*Part of the material in this chapter is reprinted by permission, A. M. Marzouk, E. Moreno-Centeno, and H. Üster, A Branch and Price Algorithm for Solving the Hamiltonian p-median Problem, INFORMS Journal on Computing, 28(4):674-686, 2016. Copyright 2016, the Institute for Operations Research and the Management Sciences, 5521 Research Park Drive, Suite 200, Catonsville, Maryland 21228 USA.

of Z. This implies that there exists a feasible solution of HpMP on G', which is comprised of p copies of G, of value pZ. The optimality of this value stems from two facts. First, no optimal solution to the HpMP on G' can contain any of the edges with length $(p+1)H$ as any such solution has a total length that is greater than pZ (since $(p+1)H$ is strictly greater than pZ). Second, when restricted to each copy of G in G' (as none of the edges connecting two copies can be selected in any optimal solution), the least length cycle in each copy is the TSP tour of value Z. Thus, by considering the p copies, the optimal solution of HpMP for a specific value of p is pZ. Next, suppose that an optimal HpMP solution on G' has a value of pZ. This implies that none of the edges with length $(p+1)H$ is selected since pZ is strictly less than $(p+1)H$. Thus, by considering only one copy of G in G', the TSP solution on this subgraph is readily available and equals to Z as G' (after deleting the edges with length $(p+1)H$) is equivalent to p disconnected copies of G. □

In this chapter, we present a B&P algorithm to solve HpMP. To this end, we start by presenting the set partitioning model for HpMP followed by formulating the pricing problem. Next, we present the algorithms used to solve the pricing problem. We end this chapter by presenting the computational results and conclusions.

2.1.1 Problem Formulation

Before presenting the set partitioning formulation for HpMP, we give a formal definition of the HpMP and the notation used. Given an **undirected** graph $G = (V, E)$ with edge cost $d_{ij} \geq 0$ for all edges $(i, j) \in E$, and a positive integer p, find p simple cycles such that each node $i \in V$ is contained in exactly one cycle and the sum of the costs of these cycles is minimum. Note that since the graph is undirected, each cycle should pass through at least three nodes.

Let x_k be a binary variable that represents whether the cycle k is in the optimal

solution. The cost of the cycle k is denoted by c_k and is calculated by summing the costs of the edges in the cycle. The coefficients a_{ik} denote which nodes are included in cycle k (i.e., a_{ik} is 1 if cycle k passes through node i, and is 0 otherwise). The set of all cycles in G is denoted by K. Note that the size of K is exponential.

HpMP can be formulated as a set partitioning problem with an additional constraint enforcing that a feasible solution must have exactly p cycles (constraint (2.1c)):

$$\textbf{(Master Problem)} \quad \text{minimize} \sum_{k \in K} c_k x_k \tag{2.1a}$$

$$\text{subject to} \sum_{k \in K} a_{ik} x_k = 1 \qquad \forall i \in V \tag{2.1b}$$

$$\sum_{k \in K} x_k = p \tag{2.1c}$$

$$x_k \in \{0, 1\} \qquad \forall k \in K \tag{2.1d}$$

Constraints (2.1b) imply that every node $i \in V$ must be covered by exactly one cycle/variable. The objective (2.1a) is to minimize the sum of the costs of the p selected cycles. As opposed to the set partitioning formulation presented in [21], the cycles in set K need not be the least cost Hamiltonian cycles; specifically K contains every simple cycle passing through at least three nodes. Henceforth, the terms *column* and *variable* are used interchangeably.

2.1.2 Contributions

The main goal in this chapter is to develop and implement a B&P algorithm to solve HpMP. To this end, this chapter includes several contributions on modeling, methodology, and computational aspects: 1) we modified the set partitioning formulation of HpMP proposed by [21]. Specifically, we removed the condition that the

columns in the set partitioning formulation be least cost Hamiltonian cycles, and thus allowed every simple cycle to be a valid column. Note that this formulation change does not affect the tightness of the LP relaxation (in [21], it is reported that the set partitioning formulation had the tightest LP bounds). However, this simple modification greatly simplifies the pricing problem, thus effectively allowing us to solve larger instances; 2) we developed a new efficient algorithm to find the shortest cycle in an undirected graph with arbitrary edge costs and no negative cycles; 3) we developed an algorithm to find the most negative cycle in an undirected graph with arbitrary edge costs; 4) computationally, the proposed algorithm for solving the HpMP outperformed the previously presented algorithms as it successfully solves instances up to 318 nodes, as opposed to other exact algorithms which only solved instances up to 100 nodes; 5) since the proof in [13] only applies to general, incomplete graphs, we refined it to establish that for every value of p, HpMP is NP-hard even when restricted to Euclidean graphs; and 6) we showed that the practical complexity of HpMP and the performance of the algorithms to solve it substantially depend on the relation between p and p_{2m} (the number of cycles in the 2-matching optimal solution). Contributions 2 and 3 are relevant here because these two problems arise when solving the pricing problem. However, these two problems are important on their own right.

The structure of this chapter is as follows: the solution methodology and the pricing problem are discussed in Section 2.2. The test instances and the computational results compared to those of the best IP model in [21] are reported in Section 2.3; finally Section 2.4 presents the conclusions.

2.2 Solution Approach

In this section, a B&P approach is presented to solve the set partitioning formulation of the HpMP. For this purpose, we first give an overview of this approach and then the details are presented in the next subsections.

This method starts by considering only a small subset, K', of the set of all simple cycles in G, K, to define a restricted master problem (RMP) from Problem 2.1. Next, the linear programming (LP) relaxation of the RMP is solved and using its dual solution, a pricing problem is constructed. As we discuss below, the pricing problem entails finding a cycle with negative reduced cost on an undirected graph with arbitrary edge weights. If no such cycle exists, then the optimal solution of the RMP's LP relaxation is also optimal for Problem 2.1. Otherwise, a new cycle (having a negative reduced cost) is added to the RMP and the RMP's LP relaxation is resolved. Branching starts when there are no cycles with negative reduced cost and the solution of the RMP's LP relaxation is fractional. These same steps are applied (on a slightly modified RMP and pricing problem) at each subsequent node in the branching tree until we get the optimal integer solution.

2.2.1 Initialization

To form an initial RMP, the first step in any B&P algorithm is to populate the initial set K' with a set of variables (i.e., simple cycles) such that the RMP has a feasible solution. In our problem, we devise the following simple approach to find an initial solution efficiently. The idea is to initialize the RMP with a set of artificial columns having an arbitrary high cost. Specifically, given $N = |V|$ and p, the first column will contain nodes $1, \ldots, \lfloor N/p \rfloor$, the second column will have nodes $\lfloor N/p \rfloor + 1, \ldots, 2\lfloor N/p \rfloor$ and so on. Note that the last column may contain a larger number of nodes. For those columns whose nodes visited in canonical order do form

a cycle in the given graph, the arbitrary high cost is replaced by the cost of the respective canonical cycle.

2.2.2 Pricing Problem

This subsection defines the pricing problem associated with the RMP. Let μ_i for $i = 1, \ldots, |V|$ be the dual variables associated to constraints (2.1b) and let μ_0 be the dual variable for constraint (2.1c). Note that since constraints (2.1b) and (2.1c) are equality constraints, all of these dual variables can be positive or negative (i.e., can be arbitrary real numbers).

In the RMP, as in Problem 2.1, each column represents a cycle, k, that passes through a set of nodes; the i^{th} element in the column is 1 if cycle k passes through node $i \in V$ and is 0 otherwise (this information is encoded in the parameter a_{ik}). This is true for the first $|V|$ elements in the column, whereas the last element (i.e., the coefficient in constraint (2.1c)) is always one. Thus, in the pricing problem, we define a new binary variable y_i, $i \in V$ such that y_i equals one if the cycle passes through node i and y_i equals zero otherwise. Cycle k also passes through a set of edges and thus a binary variable t_{ij} is defined for each edge $(i, j) \in E$ and will have a value of one if cycle k passes through edge (i, j) and a value of zero otherwise. Hence, the cost of any cycle can be written as $\sum_{(i,j)\in E} d_{ij} t_{ij}$.

Thus, the objective function in the pricing problem, which corresponds to calculating the reduced cost of a column, can be written as minimize $Z = \sum_{(i,j)\in E} d_{ij} t_{ij} - \sum_{i\in V} \mu_i y_i - \mu_0$ and the constraints of the pricing problem enforce the consistency of t_{ij} and y_i forming a simple cycle. Now, since any node in a simple cycle has exactly two adjacent edges, the objective function of the pricing problem can be rewritten as minimize $Z = \sum_{(i,j)\in E} (d_{ij} - \frac{\mu_i}{2} - \frac{\mu_j}{2}) t_{ij} - \mu_0$.

Next, we give an alternative, more natural definition of the pricing problem.

Given an **undirected** graph $G' = (V, E)$ with edge weights $w_{ij} = d_{ij} - \frac{\mu_i}{2} - \frac{\mu_j}{2}$ (note that these weights can be positive or negative) and an arbitrary real number μ_0, a column with a negative reduced cost corresponds to a cycle whose total weight minus μ_0 is negative. To solve the pricing problem, one might be tempted to ignore μ_0, find the cycle with minimum weight, and add $-\mu_0$ to the minimum weight found. Then depending on the result of this sum being negative or positive, either add the cycle to the RMP or determine that the optimal solution of the RMP's LP relaxation has been found, respectively. However, since the undirected graph G' possibly has edges with negative weights, the solution approach outlined above is not practical since finding the most negative cycle is NP-hard [3]. Therefore, Section 2.2.3 presents a practical approach to solve the pricing problem. This approach hinges on the following remarks:

a) while finding the most negative cycle is NP-hard, determining whether a given undirected graph has a negative cycle is solvable in polynomial time [49];

b) if the graph has no negative cycle, then finding the cycle with the smallest weight is also solvable in polynomial time as discussed in Section 2.2.3.1; and

c) when solving the pricing problem, one may settle for finding a negative reduced cost cycle (or determine if no such cycle exists) instead of striving to find the cycle with the most negative reduced cost.

2.2.3 Solving the Pricing Problem

As mentioned before, solving the pricing problem entails determining whether a simple cycle, C, with negative reduced cost exists, where the reduced cost of a cycle is $Z = \sum_{(i,j)\in C} w_{ij}t_{ij} - \mu_0$. Notice that the pricing problem's objective function comprises two parts: the cycle's total weight, $W = \sum_{(i,j)\in C} w_{ij}t_{ij}$, and μ_0. Based

on this observation, depending on whether G' has a negative cycle and the sign of μ_0, we consider four cases (described below) when solving this problem. In [49], two algorithms for solving the undirected negative weight cycle detection problem were presented. The first algorithm is based on b-matching while the second is based on T-join with time complexities of $O((|V|+|E|)^2 \log(|V|+|E|))$ and $O(|V|^3)$, respectively. Although the worst case complexity of the latter algorithm is better, the computational results reported in [49] showed that the b-matching algorithm performs better in practice for complete graphs with less than 400 nodes. In addition, note that our algorithms need to find the negative cycles explicitly but the T-join algorithm only detects their presence and does not identify them. For these reasons, hereafter, we will use UNWCD to refer to undirected negative weight cycle detection algorithm based on b-matching. For completeness, the pseudo-code of the UNWCD based on b-matching is given in §A.1 in the Appendix. Briefly, this algorithm works as follows: the input graph is transformed to an auxiliary graph such that detecting whether the input graph has a negative cycle is equivalent to finding the minimum weight perfect matching in the auxiliary graph. Based on the edges in this perfect matching, one can easily find one or more negative cycles or conclude that no negative cycle exists. An important property that we exploit later is that if the output of the algorithm is more than one cycle, the cycles will necessarily be node-disjoint.

Case 1: G' has no negative cycle and $\mu_0 \leq 0$. In this case, we have an optimal solution to the LP relaxation of the RMP because the least-weight cycle in G' has a nonnegative total weight and thus subtracting μ_0 cannot lead to a cycle with negative reduced cost.

Case 2: G' has a negative cycle and $\mu_0 \geq 0$. In this case, the negative-weight cycle found by UNWCD has a negative reduced cost since subtracting a nonneg-

ative μ_0 from the (negative) total weight results in a negative reduced cost. Therefore, this cycle can be added to the RMP.

Case 3: G' has no negative cycle and $\mu_0 > 0$. In this case, deciding whether there exists a cycle with negative reduced cost depends on whether there exists a cycle whose total weight, W, is less than μ_0 (i.e., $Z = W - \mu_0 < 0$). Section 2.2.3.1 presents an algorithm to determine whether G' has such a cycle; then, if such a cycle exists it is added to the RMP, otherwise an optimal solution to the RMP's LP relaxation was found.

Case 4: G' has a negative cycle and $\mu_0 < 0$. This case can be divided into two subcases:

a) The total weight of the negative cycle detected by UNWCD is less than μ_0. This implies that $Z < 0$ and the cycle is added to the RMP as it has a negative reduced cost.

b) The total weight of the negative cycle detected by UNWCD is greater than or equal to μ_0. That is, this cycle has a nonnegative reduced cost; however, as this cycle may not be the most negative cycle, we cannot conclude that we have an optimal LP solution to the RMP. Section 2.2.3.2 describes a strategy to find a cycle such that $Z < 0$ (then this cycle is added to the RMP) or determine that no such cycle exists (then we conclude that we have an optimal LP solution to the RMP).

2.2.3.1 Algorithm to Solve the Problem Arising in Case 3

Given an **undirected** graph $G' = (V, E)$ with arbitrary edge weights w_{ij} where G' has no negative cycles and given a positive number μ_0, the problem is to find a

cycle whose total weight, W, is less than μ_0 or determine that no such cycle exists. This subsection describes two algorithms to solve this problem:

Algorithm 1's strategy is to find the least-weight cycle and then compare its weight to μ_0. In [3], an algorithm to find the least-weight path between any two nodes in undirected graphs with arbitrary edge weights when the graph has no negative cycle is presented. Thus, to find the least-weight cycle, one can run the algorithm in [3] for every pair of nodes $i, j \in V$, after deleting the edge (i, j), and then add the edge weight joining nodes i and j to close the cycle. The complexity of the algorithm to find the least-weight path is $O(|V|^6)$ [3]. Thus, finding the least-weight cycle has a complexity of $O(|V|^8)$.

One may be tempted to improve the complexity from $O(|V|^8)$ to $O(|V|^7)$ by running the algorithm in [3] only for every single node i by splitting node i into two nodes. However, this strategy is incorrect because it does not guarantee that the formed cycle passes through at least three nodes.

Algorithm 2, given below, is a correct $O(|V|(|V| + |E|)^2 \log(|V| + |E|))$ algorithm to solve our problem. The main idea of this algorithm is to transform G' to a new graph G'' by incorporating μ_0 in such a way that the existence of a cycle whose weight is less than μ_0 in G' is equivalent to the existence of a negative cycle in G''.

Algorithm 2

For each node $i \in V$, perform the following operations:
Step 1: Create G''_i as an exact copy of G'.
Step 2: Subtract $0.5\mu_0$ from all the edges incident to node i in G''_i.
Step 3: Run UNWCD on G''_i
Step 4: If UNWCD finds a negative cycle in G''_i, quit and return such cycle.
If after processing all nodes, no negative cycle was found, then report that there exists no cycle whose weight is less than μ_0.

Lemma 2.2.1 establishes the correctness of Algorithm 2, and Lemma 2.2.2 gives its complexity.

Lemma 2.2.1. *There exists at least one G_i'' that contains a negative cycle if and only if the weight of the least-weight cycle in G' is less than μ_0.*

Proof. The result follows from the following observation. If a cycle C passes through node $i \in V$, then the weight of C in G_i'', $W_C(G_i'')$, equals the weight of C in G', $W_C(G')$, minus μ_0. That is, $W_C(G_i'') = W_C(G') - \mu_0$ if cycle C passes through node $i \in V$. □

Lemma 2.2.2. *The complexity of Algorithm 2 is $O(|V|(|V| + |E|)^2 \log(|V| + |E|))$.*

Proof. Algorithm 2 runs UNWCD once for every node in G'. Since UNWCD's complexity is $O((|V| + |E|)^2 \log(|V| + |E|))$ [49], the complexity of Algorithm 2 is $O(|V|(|V| + |E|)^2 \log(|V| + |E|))$. □

We conclude this subsection with the following two remarks:

Remark 1 Instead of terminating Algorithm 2 once a negative cycle is found (and adding the cycle to the RMP), one can iterate through all the nodes. Doing so enables us to add to the RMP several cycles with negative reduced costs at once; or only adding to the RMP the cycle with the most negative reduced cost. Our experimental tests show that these two modifications are less efficient than the approach in Algorithm 2.

Remark 2 Algorithm 2 can be modified to find the least cost cycle in any given undirected graph, G', with arbitrary edge costs and no negative cycles. This modification is as follows. (1) Let $M = F + \epsilon$ where F is the cost of the edge having the largest cost in G' and $\epsilon > 0$. (2) Set $\mu_0 = M|V|$ and run Algorithm 2 for each node $i \in V$. (3) Compare the costs of the cycles found (one cycle for each value

of $i \in V$) to find the least cost cycle. This modified algorithm is correct because: a) if UNWCD finds exactly one cycle in a graph, then this cycle must be the least cost cycle; b) by subtracting $0.5M|V|$ from the edges incident to node i to form G_i'', UNWCD will necessarily find a negative cycle passing through node i since the cost of the least cost cycle cannot be greater than $M|V|$; and c) in any of the resulting G_i'', UNWCD cannot find more than one cycle since G' originally had no negative cycles and the only difference between G' and any of the G_i'''s is in the costs of the edges incident to node i (i.e., all the negative cycles, if any, pass through node i in G_i'', and thus, since UNWCD's output is restricted to node-disjoint cycles, it will output the most negative cycle from these cycles—an example illustrating this property is given in Figure A.1 in Appendix A). Clearly, the complexity of the modified Algorithm 2 is still $O(|V|(|V|+|E|)^2 \log(|V|+|E|))$.

2.2.3.2 Algorithm to Solve the Problem Arising in Case 4

In case 4, we need to solve the following problem: Given an **undirected** graph $G' = (V, E)$ with arbitrary edge weights w_{ij} for every edge $(i, j) \in E$, find a negative cycle in G' whose weight is less than μ_0 or determine that none exists. The core of our algorithm to solve this problem is the integer program (IP) given in Problem (2.2). This IP selects a set of simple cycles whose sum of weights is minimum, each cycle's weight is negative and none of these cycles is contained in a given set Q. (Our algorithm uses Q to exclude the simple cycles whose sum of weights is less than μ_0 but the weight of each cycle is greater than or equal μ_0.)

$$\textbf{(IP for Case 4)} \quad \text{minimize } T = \sum_{(i,j)\in E} w_{ij} y_{ij} \tag{2.2a}$$

$$\text{subject to} \sum_{\forall (i,j)\in\delta(v)} y_{ij} = 2z_v \qquad \forall v \in V \tag{2.2b}$$

$$\sum_{\forall(i,j)\in q} y_{ij} \leq |q| - 1 \qquad \forall q \in Q \qquad (2.2c)$$

$$y_{ij} \in \{0,1\} \qquad \forall (i,j) \in E \qquad (2.2d)$$

$$z_v \in \{0,1\} \qquad \forall v \in V \qquad (2.2e)$$

In this model, the value of the binary variable y_{ij} is one if edge (i,j) is selected in the optimal solution, and is 0 otherwise. The value of the binary variable z_v is one if node $v \in V$ is included in one of the selected cycles. In the constraints (2.2b), $\delta(v)$ denotes the set of edges incident to node $v \in V$; these constraints imply that if node $v \in V$ is selected, then exactly two of the edges incident to v must be selected; whereas if node v is not selected, then none of its incident edges can be selected. Consequently, constraints (2.2b) enforce the solution to form only simple cycles. In the constraints (2.2c), $|q|$ denotes the number of edges in cycle $q \in Q$; these constraints prohibit the formation of any cycle in Q. Note that Problem (2.2) requires exponential time in the worst case.

Algorithm 3, given below, uses Problem (2.2) to find a cycle in G' whose weight is less than μ_0 or conclude that none exists:

Note that the cardinality of Q may become exponential during Algorithm 3. However, in practice the size of Q is relatively small. This is because, as shown in our computational results, for the vast majority of the instances, Algorithm 3 solves at most three IPs to find a cycle that is more negative than μ_0 or to conclude that none exists.

Lemma 2.2.3. *Algorithm 3 is correct and terminates in a finite number of steps.*

Proof. Clearly, when the algorithm reaches the stopping condition in step 3, its output is correct. The stopping condition in step 4 is correct because T^* being

Algorithm 3

Input: $G' = (V, E)$, a weight w_{ij} for all edges $(i, j) \in E$, and a negative number, μ_0.
Output: A cycle or set of cycles such that the weight of each cycle is less than μ_0 or conclude that no such cycle exists.
Step 1: Let Q be an empty set.
Step 2: Solve Problem (2.2).
Step 3: If one or more of the cycles in the optimal solution to Problem (2.2) has weight less than μ_0, then return these cycle(s) and quit. Otherwise, continue to step 4.
Step 4: If the sum of the weights of the cycles in the optimal solution of Problem (2.2), T^*, is greater than or equal to μ_0, then report that no negative cycle in G' has weight less than μ_0 and quit. Otherwise, continue to step 5.
Step 5: ($T^* < \mu_0$ but no cycle in the optimal solution has a weight less than μ_0). Update the set Q by adding to it all the cycles in the optimal solution of Problem (2.2), and go to step 2.

greater than or equal to μ_0 implies that G' does not contain a cycle whose weight is less than μ_0. This implication follows from these observations: 1) all of the cycles in Q have weight that is greater than or equal to μ_0 (see step 5), and 2) the existence of a negative cycle whose weight is less than μ_0 contradicts the optimality of T^*. Now, the operations performed in step 5 only prevent cycles whose weight is greater than or equal to μ_0 from being selected in future iterations; therefore these operations do not affect the correctness of the algorithm and thus we conclude that Algorithm 3 is correct. To see that Algorithm 3 must terminate in a finite number of steps, we first note that if Q were to contain all of the negative cycles in G', then T^* would equal 0, and thus the termination condition of step 4 would be met. Now since in each iteration, step 5 adds at least one cycle to Q and G' has a finite number of cycles, it follows that Algorithm 3 terminates in a finite number of iterations. □

Remark 3 Algorithm 3 can be modified to find the most negative cycle in undirected graph with arbitrary edge costs. The modification is as follows. First, define

Q as an empty set and let S be zero. Second, solve Problem (2.2) and add all the cycles in the optimal solution to Q. Finally, let S be the cost of the most negative cycle found so far in Q. Repeat the last two steps until either T^* is greater than or equal to S or T^* is zero (in which case, the graph contains no negative cycles except those already contained in Q). After the stopping condition is met, if S is less than zero, then it is the cost of the most negative cycle (which can easily be extracted from Q); otherwise S is equal to zero and the graph has no negative cycles.

2.2.4 Branching Strategy

In a B&P framework, after finding an optimal solution to the LP relaxation of the RMP at a branching-tree node, if at least one of the variables (cycles) is not integer, we have to apply branching. For our problem, a natural choice is to branch on the cycles, i.e., pick a cycle k having a fractional x_k value and create two branches: one with $x_k = 1$ and the other with $x_k = 0$. We did not adopt this approach because in the zero branches, the efficient algorithms for solving cases 2 and 3 (described in Section 2.2.3) will no longer be efficient; specifically, the problems that would need to be solved in cases 2 and 3 on any branching-tree node preceded by one or more zero branches become NP-hard.

Therefore, instead of branching on the cycles, we branch on the edges. This branching rule is inspired by the one used by [16] to solve the crew scheduling problem which in turn is a specialization of the branching rule proposed by [40] to solve the same problem. The following lemma proves the correctness of our branching rule.

Lemma 2.2.4. *If the solution to the RMP's LP relaxation is fractional, then there exists an edge (u, v) such that $0 < \sum_{j \in S(u,v)} x_j < 1$ where $S(u, v) = \{j \mid x_j > 0$ and cycle j has edge $(u, v)\}$.*

Proof. Let cycle t be a cycle with fractional x_t value. Assume that cycle t has k

nodes, and refer as n_i to the i^{th} node of cycle t; specifically cycle t is $n_1 n_2 \dots n_k n_{k+1}$ where $n_{k+1} = n_1$. First, since x_t is fractional, we have $\sum_{j \in S(n_i, n_{i+1})} x_j > 0$ for every $i = 1, \dots, k$. Thus, we only need to prove that $\sum_{j \in S(n_i, n_{i+1})} x_j < 1$ for at least one $i = 1, \dots, k$. For the sake of contradiction, assume that

$$\sum_{j \in S(n_i, n_{i+1})} x_j = 1, \forall i = 1, \dots, k. \tag{2.3}$$

Define the set of cycles $R(u) = \{j \mid x_j > 0 \text{ and cycle } j \text{ has node } u\}$. From constraint (2.1a) in the RMP, we have

$$\sum_{j \in R(n_i)} x_j = 1, \forall i = 1, \dots, k. \tag{2.4}$$

Equation (2.4) can be rewritten as

$$\sum_{j \in R(n_i) \cap S(n_i, n_{i+1})} x_j + \sum_{j \in R(n_i) \setminus S(n_i, n_{i+1})} x_j = 1, \forall i = 1, \dots, k. \tag{2.5}$$

As $S(n_i, n_{i+1}) \subseteq R(n_i)$, equation (2.5) is equivalent to

$$\sum_{j \in S(n_i, n_{i+1})} x_j + \sum_{j \in R(n_i) \setminus S(n_i, n_{i+1})} x_j = 1, \forall i = 1, \dots, k. \tag{2.6}$$

The first term in equation (2.6) equals 1 by our assumption (equation (2.3)), therefore we have that $\sum_{j \in R(n_i) \setminus S(n_i, n_{i+1})} x_j = 0, \forall i = 1, \dots, k$. This, in turn, implies that any cycle j, such that $x_j > 0$, passing through node n_i must pass through edge (n_i, n_{i+1}); equivalently $R(n_i) \subseteq S(n_i, n_{i+1})$. Since this is true for every node n_i, $i = 1, \dots, k$, we conclude that

$$R(n_i) = S(n_i, n_{i+1}), \forall i = 1, \dots, k. \tag{2.7}$$

A similar argument shows that

$$R(n_i) = S(n_{i-1}, n_i), \forall i = 2, \ldots, k+1. \tag{2.8}$$

From equations (2.7) and (2.8), we conclude that

$$S(n_{i-1}, n_i) = S(n_i, n_{i+1}), \forall i = 2, \ldots, k. \tag{2.9}$$

On the other hand, by the uniqueness of cycle t (the optimal basis of the RMP's LP relaxation has no duplicate cycles), t is the only cycle that passes through edges $(n_1, n_2), (n_2, n_3), \ldots, (n_k, n_{k+1})$ (i.e., $\cap_{i=1}^{k} S(n_i, n_{i+1}) = \{t\}$). This fact and equation (2.9) imply that $S(n_i, n_{i+1}) = \{t\}$, $\forall i = 1, \ldots, k$. Thus, equation (2.3) can be written as $\sum_{j \in \{t\}} x_j = 1$, $\forall i = 1, \ldots, k$; implying that $x_t = 1$, which contradicts the fractionality of x_t. □

Based on this lemma, after selecting an edge (u, v) such that $0 < \sum_{j \in S(u,v)} x_j < 1$, we branch on edge (u, v) as follows:

0-branch In this branch, we need to add the constraint $\sum_{j \in \gamma(u,v)} x_j \leq 0$ where $\gamma(u, v)$ contains all cycles that include edge (u, v). This is equivalent to ensuring that the solutions found on this branching node and its successors do not contain a cycle passing through edge (u, v). Instead of adding this constraint explicitly, it can be implicitly enforced by: 1) deleting all cycles in the RMP that have edge (u, v), and 2) removing the edge (u, v) from graph G so that the pricing problem does not generate any new cycles having edge (u, v).

1-branch In this branch we need to add the constraint $\sum_{j \in \gamma(u,v)} x_j \geq 1$. This is equivalent to ensuring that the solutions found on this branching node and its successors must contain a cycle passing through edge (u, v). In contrast to the 0-branch, in the 1-branch, we add explicitly this constraint to the RMP.

Therefore, we also need to modify the pricing problem accordingly as follows: let λ_{uv} be the dual variable associated with the newly added constraint. Then, the objective function of the pricing problem becomes the minimization of $Z = \sum_{ij \in E, ij \neq uv}(d_{ij} - \frac{\mu_i}{2} - \frac{\mu_j}{2})t_{ij} + (d_{uv} - \frac{\mu_u}{2} - \frac{\mu_v}{2} - \lambda_{uv})t_{uv} - \mu_0$ which can alternatively be written as the minimization of $Z = \sum_{ij \in E}(d_{ij} - \frac{\mu_i}{2} - \frac{\mu_j}{2})t_{ij} - \lambda_{uv}t_{uv} - \mu_0$. Therefore, the only difference of the pricing problem in this 1-branch with respect to the root node's pricing problem (described in Section 2.2.2) is that the weight of edge (u, v) is $d_{uv} - \frac{\mu_u}{2} - \frac{\mu_v}{2} - \lambda_{uv}$. As such, the algorithms described in Sections 2.2.3.1 and 2.2.3.2 can also be used to solve the pricing problems arising during the branching strategy.

We conclude this section with the following remark: in our computational results we found that using the best-bound strategy (i.e., branching on the node with the lowest RMP's LP relaxation objective value) outperformed the breadth-first and depth-first strategies, especially in large instances.

2.3 Computational Results

In this section, we present the computational results of our B&P algorithm. The algorithm was tested on 30 instances from the TSPLIB available from `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/`. Selected instances are complete graphs with number of nodes ranging from 17 nodes to 318 nodes. Following the convention adopted in the TSP literature, the edge costs are rounded to the nearest integer. We compared the performance of our B&P algorithm to that of solving the first IP model given in [21].* We chose to compare against this model because computational results in [21] showed that this was the most efficient

*Since the original code was not available, we used our own implementation, in which, a sequence of IP models was solved with successive additions of integral cuts to enforce the formation of exactly p cycles.

and effective model among the seven models they tested. For completeness, this model is presented in §A.2 in the Appendix, and hereafter, we refer to this model as the natural variable space model (NVM). For each instance, both algorithms were tested for all possible values of the number of required cycles, p.

Both algorithms were run on a computer with an Intel Core i7 processor and 32 GB of memory. Implementations were coded and compiled on Visual Studio C++ using standard template library and standard subroutines. Linear and integer programs were solved using CPLEX 12.4 invoked in C++ using Concert Technology. A time limit of one hour is set in all our tests.

As previously discussed, in our B&P implementation we detect negative weight cycles in undirected graphs using the b-matching-based algorithm by [49]. This algorithm performed better than the T-join-based approach for our test instances (i.e., complete graphs with less than 400 nodes). We solved the matching problem using blossomV which is one of its fastest implementations. BlossomV was developed by [28] and is available online at `http://pub.ist.ac.at/~vnk/software.html`.

In the remainder of this section, the computational results comparing the performances of B&P and NVM approaches are organized as follows. Section 2.3.1 presents the computational times or optimality gaps at the one hour time limit for small and medium size TSP instances with up to 127 nodes. Section 2.3.2 compares the quality of the root node LP lower bounds obtained by B&P to that of NVM. Section 2.3.3 summarizes several performance parameters of the B&P algorithm and gives an overall comparison of B&P and NVM. Encouraged by the performance of the B&P algorithm reported in Sections 2.3.1 to 2.3.3, we further tested its performance on larger instances with up to 318 nodes. The computational times (or optimality gaps) are reported in Section 2.3.4.

2.3.1 Comparison of B&P and NVM Computational Times

Tables 2.1, 2.2, and 2.3 presents the total computational times (in seconds) for our B&P algorithm and for NVM. Specifically, Table 2.1 compares the performances of B&P and NVM for small size instances. Tables 2.2 and 2.3 compare the performance of B&P and NVM for medium size instances when p is small (i.e., $p \leq 20$), and p is large (i.e., $p > 20$), respectively.

In Tables 2.1-2.3, the first column in the table has the graph name, and the top row contains the p value (i.e., the required number of cycles). In all of the tables (both, in this chapter and in the appendix), we adopt the following conventions: (1) The entries for the p value corresponding to the number of cycles in the minimum weight 2-matching problem, p_{2m}, are marked with a bullet ($\bullet$). For example, the optimal solution of the minimum weight 2-matching problem in graph *swiss42* contains 7 cycles; (2) italic red numbers signify the instances in which NVM outperformed our B&P algorithm; (3) blue bold numbers signify the instances in which our B&P algorithm outperformed NVM; and (4) empty cells indicate that the instance is infeasible (i.e., the instances in which $p > \frac{|V|}{3}$ for complete graphs).

In Tables 2.1-2.3, whenever an instance was not solved within the time limit, instead of reporting its runtime, we report the optimality gap attained at the one hour time limit. The optimality gap is calculated as $100 \times \frac{Z_{IP}-Z_{LB}}{Z_{IP}}\%$ where Z_{IP} is the value of the best feasible solution found and Z_{LB} is the value of a valid lower bound. In some instances, B&P failed to solve the LP relaxation at the root node to optimality within the one hour time limit. In these cases, the minimum weight 2-matching solution value was used as a lower bound (this is valid as it is a relaxation of the HpMP). Finally, for those instances where B&P did not find an optimal solution within the time limit, we obtained a feasible solution by solving the *integral* restricted

master problem to optimality. Since the number of generated columns was small, the restricted master problem was solved quickly using CPLEX, specifically, for all our instances, it took less than 5 seconds.

The results from Tables 2.1-2.3 show a clear correlation between the number of cycles in the 2-matching solution and the performance of the B&P and NVM approaches. In general, we observe that the B&P algorithm outperforms NVM for p values that are greater than $p_{2m} + 3$, whereas NVM performs better whenever p is less than or equal to $p_{2m} + 3$. Based on the results in Tables 2.1-2.3, we divide the instances based on varying p values into three subsets and make the following observations:

- **Instances with $p < p_{2m} - 3$:** These instances are challenging for both B&P and NVM. But, NVM performs better than B&P in terms of the number of instances solved to optimality and in the average optimality gap for the unsolved instances. Specifically, from these 107 instances, NVM succeeded in finding the optimal solution for 67 instances with an average running time of 138 seconds. On the other hand, B&P found the optimal solution for only 5 instances in 468 seconds, on average. For the unsolved instances by NVM, the average optimality gap was 6.1%, whereas B&P's average optimality gap for its unsolved instances was 64%. The performance of the B&P algorithm deteriorates as p approaches one. For instances with optimality gaps greater than 50%, the B&P algorithm failed to solve even the root node to optimality after the one hour time limit. Thus, the 2-matching solution value was used as lower bound. Moreover, in most of these cases, B&P also failed to find a feasible solution other than the initial solution provided to start the column generation. A explanation of the reason of this poor performance is provided

in subsection 2.3.3.

- **Instances with $p_{2m} - 3 \leq p \leq p_{2m} + 3$:** There are 124 such instances; on these, the performance of NVM is excellent. Specifically, NVM solved 120 instances to optimality with an average computational time of 47 seconds, whereas the B&P solved only 66 to optimality with an average time of 433 seconds. In contrast, B&P outperformed NVM in terms of the average optimality gap for the respective unsolved instances. Specifically, for the unsolved instances by NVM, the average optimality gap was 32%, whereas the gap was 23% for the unsolved instances by B&P. Moreover, interestingly, for the four instances that NVM failed to solve, B&P solved three of them to optimality and in the remaining one B&P's gap was 0.1% whereas NVM's gap was 19%. The overall good performance of NVM in these instances is because NVM added only a small number of cuts to find the optimal solutions of these instances—the reason behind this phenomenon is explained in Section 2.3.3.

- **Instances with $p > p_{2m} + 3$:** There are 250 such instances; on these, B&P outperforms NVM in terms of both the computational time for the solved instances and the optimality gaps for the unsolved ones. Specifically, B&P found the optimal solution for 201 instances with average computational time of 508 seconds, whereas NVM found the optimal solutions for only 42 instances with average computational time of 323 seconds. Moreover, if B&P failed to find an optimal solution of an instance within the time limit, it provided a very good feasible solution. Specifically, the average optimality gap for the unsolved instances by B&P was 2%, whereas the average optimality gap for the unsolved instances by NVM was 38%. Again, Section 2.3.3 explains the reason behind B&P's good performance in these instances.

Ins.	Alg./p	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
gr17	B&P		21	16	5	•															
	NVM		*1*	*0*	*0*	•															
gr21	B&P	•	92	12	3	11	3	**5**													
	NVM	•	*0*	*0*	*0*	*1*	*1*	567													
gr24	B&P		14	•	3	4	26	**18**	**5**												
	NVM		*0*	•	*0*	*0*	*12*	40	522												
fri26	B&P		34	14	17	23	5	•	2												
	NVM		*0*	*0*	*1*	*0*	*0*	•	*0*												
swiss42	B&P		1848	2.9%	1803	684	509	•	26	15	6	5	**24**	**20**	**30**						
	NVM		*3*	*1*	*0*	*0*	*1*	•	*0*	*1*	*2*	*2*	44	2387	43%						
dantzig42	B&P		6.1%	278	167	325	310	294	•	4	19	14	**22**	**1305**	**4%**						
	NVM		*5*	*1*	*0*	*0*	*0*	*1*	•	*0*	*2*	*11*	102	69%	73%						
gr48	B&P		98%	1.3%	1.2%	3303	•	190	**218**	**162**	**77**	**99**	**23**	**11**	**11**	**113**	**196**				
	NVM		*11*	*2*	*0*	*0*	•	*6*	226	19%	18%	22%	25%	31%	37%	33%	40%				
hk48	B&P		85%	975	2029	107	•	25	10	75	10	**78**	**61**	**38**	**15**	**14**	**206**				
	NVM		*3*	*0*	*1*	*0*	•	*0*	*1*	*4*	*2*	129	2280	52%	57%	59%	62%				
eil51	B&P		98%	•	0.3%	381	213	96	480	**67**	**228**	**38**	**242**	**89**	**15**	**13**	**16**	**804**			
	NVM		*4*	•	*0*	*0*	*1*	*1*	*237*	397	20%	20%	24%	29%	34%	38%	46%	45%			
berlin52	B&P		78%	79%	1.7%	3508	540	•	32	274	32	12	**64**	**39**	**60**	**60**	**235**	**311**			
	NVM		*1*	*0*	*1*	*1*	*0*	•	*1*	*24*	*2*	*2*	67	2082	54%	55%	55%	54%			
brazil58	B&P		89%	89%	89%	90%	2.7%	1.4%	1%	403	0.1%	18	•	9	**19**	**50**	**91**	**305**	**223**	**198**	
	NVM		*30*	*48*	*158*	*56*	*8*	*8*	*4*	*2*	*1*	*0*	•	*1*	104	65%	66%	69%	70%	74%	

Table 2.1: Computational times (in sec) for the small TSPLIB instances (all feasible values of p).

Ins.	Alg./p	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
st70	B&P	99%	99%	4%	99%	2.3%	1.6%	1.4%	1.4%	3347	214	103	•	14	10	12	37	36	**116**	**94**
	NVM	*67*	*72*	*95*	*78*	*15*	*7*	*5*	*4*	*0*	*0*	*0*	•	*1*	*2*	*2*	*4*	*35*	26%	25%
eil76	B&P	98%	•	0.1%	1044	425	141	0.2%	35	77	63	**92**	**166**	**66**	**124**	**256**	**70**	**197**	**79**	**1005**
	NVM	*1*	•	*0*	*1*	*4*	*14*	*69*	*25*	*64*	*57*	404	19%	19%	19%	21%	26%	27%	28%	31%
pr76	B&P	61%	60%	62%	65%	66%	2.3%	•	0.1%	0.1%	**2719**	**0.1%**	**0.1%**	**0.3%**	**0.6%**	**146**	**190**	**108**	**184**	**29**
	NVM	*5%*	*6%*	*353*	*72*	*6*	*1*	•	*82*	*2601*	24%	30%	30%	35%	36%	38%	38%	41%	43%	44%
gr96	B&P	56%	54%	58%	60%	47%	62%	61%	0.3%	•	0.1%	0.1%	0.8%	**350**	**0.1%**	**0.2%**	**1.7%**	**3%**	**501**	**405**
	NVM	*210*	*56*	*26*	*40*	*16*	*10*	*3*	*1*	•	*22*	*492*	*221*	446	24%	25%	27%	27%	29%	29%
rat99	B&P	96%	96%	96%	•	96%	96%	201	168	146	350	**744**	**467**	**423**	**925**	**459**	**1932**	**681**	**484**	**833**
	NVM	*3*	*0*	*2*	•	*1*	*0*	*1*	*2*	*4*	*60*	13%	14%	14%	14%	14%	15%	15%	19%	17%
rd100	B&P	99%	99%	99%	99%	99%	99%	2.1%	1.8%	1.8%	1.3%	3309	1900	•	1178	162	148	**260**	**3529**	**0.4%**
	NVM	*2%*	*3%*	*5%*	*7%*	*57*	*67*	*134*	*17*	*9*	*8*	*4*	*1*	•	*1*	*4*	*61*	1104	26%	31%
kroA100	B&P	93%	93%	93%	93%	93%	93%	9.5%	3.4%	2.9%	0.8%	1.5%	•	0.4%	1.3%	0.7%	101	209	**331**	**427**
	NVM	*3%*	*5%*	*7%*	*10%*	*9%*	*8%*	*859*	*118*	*55*	*5*	*0*	•	*21*	*18*	*1417*	*7*	*125*	25%	25%
kroB100	B&P	91%	91%	91%	91%	91%	91%	91%	91%	2.9%	3.1%	0.6%	0.6%	0.8%	1.3%	0.2%	1.1%	0.1%	0.1%	•
	NVM	*5%*	*5%*	*4%*	*5%*	*5%*	*7%*	*2813*	*363*	*40*	*12*	*6*	*12*	*17*	*8*	*4*	*3*	*1*	*1*	•
kroD100	B&P	92%	92%	92%	92%	92%	92%	92%	6.4%	92%	2%	5.2%	92%	•	0.2%	0.1%	28	**42**	**559**	**876**
	NVM	*2%*	*7%*	*119*	*38*	*29*	*12*	*2*	*1*	*1*	*1*	*1*	*3*	•	*4*	*2*	*14*	240	19%	19%
lin105	B&P	98%	98%	98%	98%	98%	98%	97%	98%	12%	**6.3%**	**1.4%**	1.4%	1.2%	0.9%	0.3%	0.9%	3.1%	0.8%	•
	NVM	*10%*	*11%*	*7%*	*3%*	*3%*	*3%*	*5%*	*8%*	*10%*	11%	14%	*1530*	*168*	*6*	*5*	*1*	*2*	*5*	•
gr120	B&P	99%	99%	99%	99%	99%	99%	99%	99%	99%	99%	99%	99%	1.3%	•	0.4%	0.1%	**0.1%**	**542**	**1062**
	NVM	*2%*	*4%*	*4%*	*5%*	*7%*	*7%*	*9%*	*186*	*80*	*10*	*16*	*0*	*1*	•	*4*	*32*	19%	19%	20%
bier127	B&P	81%	81%	81%	81%	81%	81%	82%	81%	0.5%	0.5%	•	0.1%	0.1%	0.1%	86	262	**177**	**291**	**590**
	NVM	*4%*	*6%*	*804*	*140*	*160*	*31*	*14*	*3*	*1*	*3*	•	*1*	*1*	*2*	*1*	*29*	1633	36%	39%

Table 2.2: Computational times (in sec) for the medium TSPLIB instances and smaller values of p.

Ins	Alg./p	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
st70	B&P	**456**	**2348**	**2.1%**																	
	NVM	26%	29%	37%																	
eil76	B&P	**37**	**34**	**31**	**1021**	**1140**															
	NVM	31%	34%	39%	42%	45%															
pr76	B&P	**38**	**30**	**169**	**829**	**359**															
	NVM	45%	46%	46%	48%	47%															
gr96	B&P	**483**	**3488**	**0.2%**	**0.2%**	**0.1%**	**1667**	**1809**	**0.4%**	**1.6%**	**2.6%**	**5%**	**4.5%**								
	NVM	31%	31%	31%	31%	37%	37%	39%	49%	51%	55%	61%	67%								
rat99	B&P	**135**	**956**	**509**	**445**	**533**	**415**	**740**	**1009**	**1045**	**1199**	**377**	**3062**	**0.5%**							
	NVM	16%	16%	19%	21%	22%	24%	24%	24%	25%	26%	31%	37%	57%							
rd100	B&P	**705**	**0.1%**	**252**	**164**	**51**	**349**	**456**	**590**	**486**	**306**	**0.1%**	**0.1%**	**3.1%**							
	NVM	31%	36%	36%	41%	43%	45%	44%	48%	52%	53%	57%	59%	63%							
kroA100	B&P	**112**	**809**	**1140**	**545**	**129**	**325**	**1444**	**1428**	**1355**	**2500**	**0.5%**	**2.4%**	**2.7%**							
	NVM	27%	26%	25%	24%	23%	19%	27%	27%	31%	34%	43%	49%	51%							
kroB100	B&P	1.5%	0.1%	0.1%	57	109	**0.2%**	**138**	**48**	**48**	**124**	**883**	**3483**	**2.8%**							
	NVM	*11*	*2*	*1*	*1*	*45*	20%	24%	24%	25%	30%	43%	47%	53%							
kroD100	B&P	**884**	**657**	**560**	**278**	**182**	**223**	**336**	**3594**	**905**	**1248**	**1887**	**0.5%**	**1.6%**							
	NVM	22%	24%	25%	25%	31%	36%	36%	35%	36%	43%	48%	49%	53%							
lin105	B&P	689	726	588	733	**0.3%**	147	**43**	**1863**	**1022**	**0.4%**	**788**	**0.4%**	**1.5%**	**3.4%**	**4.7%**					
	NVM	*2*	*3*	*17*	*99*	39%	*111*	136	46%	47%	48%	47%	47%	49%	55%	58%					
gr120	B&P	**110**	**97**	**256**	**1919**	**134**	**125**	**764**	**2229**	**1004**	**481**	**602**	**883**	**562**	**87**	**108**	**1677**	**1424**	**0.1%**	**0.7%**	**2.1%**
	NVM	22%	21%	22%	25%	25%	24%	26%	30%	31%	32%	33%	33%	36%	37%	44%	48%	50%	57%	62%	62%
bier127	B&P	**156**	**476**	**95**	**104**	**232**	**101**	**125**	**101**	**218**	**1374**	**826**	**432**	**0.1%**	**2.3%**	**13%**	**11%**	**2.1%**	**2.2%**	**5.3%**	**0.2%**
	NVM	40%	39%	43%	43%	46%	47%	48%	50%	52%	52%	55%	56%	57%	58%	58%	58%	61%	63%	64%	64%

Table 2.3: Computational times (in sec) for the medium TSPLIB instances and larger values of p.

2.3.2 LP Lower Bound Comparison for B&P and NVM Based on Root Solutions

This subsection presents a comparison of the computational times and the quality of the optimal solutions of the LP relaxation at the root node for B&P and NVM. The comparison for the small size instances is shown in Table 2.4; while Tables 2.5, and 2.6 present the results for medium size instances when $p \leq 20$ and $p > 20$, respectively.

In Tables 2.4-2.6, the root gap percentage is defined as $100 \times \frac{Z_{BFS}-Z_{LP}}{Z_{BFS}}\%$, where Z_{BFS} represents the best feasible solution found, as obtained in the previous subsection, by any of the two algorithms, and Z_{LP} represents the optimal value of the LP relaxation at the root node of the tested algorithm. We decided to use Z_{BFS} for a better comparison of the tightness of the LP relaxation of both algorithms since this prevents any effect of the upper bound found by a specific algorithm on the root gap percentage. However, using Z_{BFS} makes the results in this subsection incomparable to these given in the previous subsection. As before, if column generation failed to solve the root node to optimality within the time limit, Z_{LP} is set to be the value of the minimum weight 2-matching problem (which is a valid lower bound). It is important to note that, to calculate the LP lower bounds at the root node for NVM, we employed the separation algorithms presented in [21]. Since the original code is not available, for easiness of implementation, we implemented the root node algorithm in Matlab. Since this Matlab implementation solved all of the root nodes in less than 2 seconds, using Matlab instead of C++ had no effect on the algorithm performance.

Analogously to Section 2.3.1, we divide the instances based on varying p values into three subsets and make the following observations:

- **Instances with $p < p_{2m} - 3$:** There are 107 such instances; for these, B&P

provides better LP lower bounds for 78 instances, whereas NVM provides better LP bounds for 29 instances. The average root gap percentage was 3.82% for B&P with average computational time of 3397 seconds, whereas the average root gap was 4.28% for NVM with average time is less than one second.

- **Instances with $p_{2m} - 3 \leq p \leq p_{2m} + 3$:** There are 124 such instances; for these, B&P provides better LP lower bounds for 120 instances, whereas NVM provides better LP bounds for 2 instances. The average root gap percentage was 0.27% for B&P with average computational time of 1289 seconds, whereas the average root gap was 1.24% for NVM with average time of less than one second.

- **Instances with $p > p_{2m} + 3$:** There are 250 such instances; for all of these instances, B&P provides better LP lower bounds than those provided by NVM. The average root gap percentage was 0.74% for B&P with average computational time of 76 seconds, whereas the average root gap was 4.5% for NVM with average time of less than 2 seconds.

In summary, the results in Tables 2.4-2.6 clearly show that the LP relaxation at the root node is tighter when using the B&P algorithm in 448 out of the 481 instances. Moreover, in the 31 instances where NVM's root gap outperformed B&P's root gap, it was because B&P was not able to solve the root node's LP relaxation to optimality (thus the minimum-weight 2-matching value was used as a lower bound). The average root gap percentage (for the 481 runs) was 1.3% when using the B&P algorithm, whereas this gap increased to 3.63% when using the NVM. In contrast, NVM was much faster in solving the root node.

Ins	Alg./p	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
gr17	BP root %	**0**	**0**	**0**	•														
	BP root time	17	13	4	•														
	NVM root %	4.24	1.02	*0*	•														
	NVM root time	*1*	*1*	*0*	•														
gr21	BP root %	**2.04**	**1.34**	**0**	**1.36**	**0**	**0**												
	BP root time	6	2	2	4	2	5												
	NVM root %	2.36	2.39	1.78	3.77	2.67	8.39												
	NVM root time	*0*	*0*	*1*	*1*	*1*	*1*												
gr24	BP root %	**0**	•	**0**	**0**	**0.55**	**0.39**	**0**											
	BP root time	14	•	3	4	4	3	5											
	NVM root %	0.61	•	0.20	1.01	3.28	4.93	7.02											
	NVM root time	*0*	•	*0*	*1*	*1*	*1*	*1*											
fri26	BP root %	**0**	**0**	**0.77**	**0.11**	**0**	•	**0**											
	BP root time	23	11	8	5	5	•	2											
	NVM root %	1.48	1.54	2.76	1.46	0.68	•	0.56											
	NVM root time	*0*	*0*	*0*	*0*	*0*	•	*1*											
swiss42	BP root %	**0**	2.86	**0.02**	**0.33**	**0.49**	•	**0.08**	**0.08**	**0**	**0**	**0.08**	**0.08**	**0.04**					
	BP root time	1331	431	489	38	10	•	9	8	6	5	10	10	16					
	NVM root %	2.39	*2.56*	1.42	1.18	1.34	•	1.34	1.66	1.9	2.29	3.3	4.37	6					
	NVM root time	*0*	*0*	*0*	*0*	*0*	•	*1*	*1*	*1*	*1*	*1*	*1*	*1*					
dantzig42	BP root %	**2.27**	**0**	**0.06**	**0.12**	**0.03**	**0.86**	•	**0**	**0.15**	**0.10**	**0.15**	**3.38**	**11.85**					
	BP root time	3600	278	40	21	18	22	•	4	9	11	9	10	19					
	NVM root %	3.03	1.08	1.08	1.08	0.93	1.54	•	1.08	1.99	2.58	3.46	8.56	19.47					
	NVM root time	*0*	*0*	*1*	*1*	*1*	*1*	•	*1*	*1*	*1*	*1*	*1*	*1*					
gr48	BP root %	**6.92**	**0**	**1.09**	**0.76**	•	**1.01**	**1.16**	**1.11**	**0.78**	**0.67**	**0.11**	**0**	**0**	**0.55**	**1.12**			
	BP root time	3600	3402	2487	74	•	12	15	13	14	17	17	11	11	19	25			
	NVM root %	2.24	1.77	1.47	0.77	•	1.57	2.41	3.19	3.73	4.51	4.83	5.58	6.86	9.42	12.42			
	NVM root time	*0*	*0*	*0*	*0*	•	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*			
hk48	BP root %	1.43	0.76	**0.19**	**0**	•	**0.07**	**0**	**0.34**	**0**	**0.37**	**0.38**	**0.13**	**0**	**0**	**1.96**			
	BP root time	3600	3600	3487	3303	•	12	10	13	10	11	13	14	15	14	24			
	NVM root %	*0.77*	*0.64*	0.65	0.20	•	0.15	0.16	0.84	0.85	1.69	2.18	2.67	3.3	4.25	8.03			
	NVM root time	*0*	*1*	*0*	*0*	•	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*			
eil51	BP root %	**0.95**	•	**0.71**	**0.48**	**0.42**	**0.27**	**0.56**	**0.31**	**0.47**	**0.13**	**0.66**	**0.37**	**0**	**0**	**0**	**2.76**		
	BP root time	3600	•	37	26	27	22	16	23	19	14	20	25	15	13	16	34		
	NVM root %	1.06	•	0.83	0.83	1.07	1.3	2	2.23	2.91	3.14	4.25	4.69	5.13	6.19	7.24	11.57		
	NVM root time	*0*	•	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*		
berlin52	BP root %	0.6	0.04	**0.28**	**0.19**	**0.06**	•	**0.06**	**0.42**	**0.07**	**0**	**0.11**	**0.10**	**0.10**	**0.32**	**0.66**	**0.85**		
	BP root time	3600	3600	3600	68	47	•	17	15	14	12	11	16	17	21	41	55		
	NVM root %	*0.28*	*0*	0.29	0.24	0.08	•	0.25	0.78	0.6	0.69	1.3	1.86	2.45	3.58	5.76	8.15		
	NVM root time	*1*	*1*	*0*	*0*	*0*	•	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*		
brazil58	BP root %	6.33	5.97	**5.12**	**3.09**	**1.85**	**1.42**	**1.01**	**0**	**0.11**	**0**	•	**0**	**0**	**0.16**	**0.39**	**0.90**	**1.20**	**2.01**
	BP root time	3600	3600	3600	3600	3600	3600	3600	403	446	18	•	9	19	25	13	14	22	33
	NVM root %	*2.12*	*5.45*	5.34	3.52	2.44	2.11	1.85	1.45	1.27	0.87	•	1.01	1.53	2.3	3.14	4.35	5.64	7.68
	NVM root time	*1*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*1*	•	*3*	*4*	*4*	*5*	*6*	*7*	*8*

Table 2.4: Performance of B&P and NVM at the respective root node for the small TSPLIB instances

Ins	Alg./p	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
st70	BP root %	5.16	4.14	3.55	2.95	2.19	**1.53**	**1.42**	**1.11**	**0**	**0**	**0**	•	**0**	**0.32**	**0**	**0.11**	**0.04**	**0.16**	**0.16**
	BP root time	3600	3600	3600	3600	3600	3600	3600	3600	3347	168	103	•	14	22	12	19	26	23	26
	NVM root %	*3.31*	*3.21*	*2.96*	*2.62*	*2.06*	1.54	1.50	1.25	0.52	0.40	0.24	•	0.24	0.72	0.56	1.03	1.34	1.97	2.58
	NVM root time	*1*	*1*	*1*	*1*	*2*	*1*	*1*	*1*	*1*	*1*	*1*	•	*1*	*1*	*1*	*1*	*1*	*1*	*1*
eil76	BP root %	0.37	•	**0**	**0.04**	**0.07**	**0.11**	**0.15**	**0**	**0**	**0**	**0**	**0.14**	**0.05**	**0.09**	**0.27**	**0.03**	**0.26**	**0.23**	**0.20**
	BP root time	3600	•	311	85	64	64	58	35	57	45	57	76	47	43	42	49	59	44	44
	NVM root %	*0.19*	•	0	0.18	0.34	0.53	0.68	0.65	0.8	0.96	1.11	1.43	1.56	1.88	2.37	2.47	3.1	3.55	3.97
	NVM root time	*0*	•	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
pr76	BP root %	**7.15**	**7.91**	**2.74**	**2.35**	**0.41**	**0.19**	•	**0.36**	**0.44**	**0.54**	**0.67**	**0.71**	**0.89**	**1.06**	**0.18**	**0.28**	**0.17**	**0.14**	**0**
	BP root time	3600	3600	3600	3600	2498	2987	•	58	43	53	36	44	60	77	65	63	59	32	29
	NVM root %	8.79	9.59	4.58	4.29	2.84	2.38	•	2.45	2.74	3.07	3.46	3.79	4.24	4.69	4.13	4.51	4.83	5.25	5.57
	NVM root time	*1*	*1*	*0*	*0*	*0*	*1*	•	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
gr96	BP root %	2.68	**1.59**	**1.11**	**0.92**	**0.66**	**0.37**	**0.19**	**0.01**	•	**0.12**	**0.26**	**0.19**	**0.18**	**0.47**	**0.67**	**1.98**	**3.12**	**0.13**	**0.06**
	BP root time	3600	3600	3600	3600	3600	3600	3600	3600	•	126	139	85	111	108	86	94	85	86	80
	NVM root %	*2.59*	1.7	1.47	1.37	1.17	0.93	0.79	0.64	•	0.75	0.96	0.99	1.09	1.51	1.84	3.26	4.54	1.77	1.87
	NVM root time	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	•	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
rat99	BP root %	**0.33**	**0.08**	**0.17**	•	**0.25**	**0.17**	**0.07**	**0.06**	**0.02**	**0.04**	**0.23**	**0.21**	**0.19**	**0.24**	**0.20**	**0.31**	**0.26**	**0.19**	**0.17**
	BP root time	3600	3600	3600	•	272	134	109	107	99	111	86	98	82	80	108	115	85	91	105
	NVM root %	0.58	0.42	0.50	•	0.42	0.42	0.42	0.50	0.58	0.75	1.16	1.48	1.8	2.2	2.52	3	3.31	3.62	4.01
	NVM root time	*0*	*0*	*0*	•	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
rd100	BP root %	**3.7**	**4.41**	**5.84**	**8.5**	**1.19**	**1.06**	**1.04**	**0.57**	**0.29**	**0.16**	**0**	**0.01**	•	**0**	**0**	**0.01**	**0.03**	**0.19**	**0.53**
	BP root time	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	1145	2415	•	349	82	104	70	71	93
	NVM root %	5.26	6.51	7.91	10.51	2.93	2.92	3.02	2.68	2.47	2.36	2.26	2.21	•	2.21	2.23	2.29	2.39	2.63	3.04
	NVM root time	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	•	*0*	*1*	*1*	*1*	*1*	*1*
kroa100	BP root %	7.28	8.75	10.23	12.88	11.77	9.94	**3.00**	**2.23**	**1.67**	**0.85**	**0.29**	•	**0.25**	**0.20**	**0.69**	**0.01**	**0.10**	**0.26**	**0.18**
	BP root time	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	•	2179	62	70	55	72	93	54
	NVM root %	*4.92*	*7.19*	*8.78*	*12.13*	*11.26*	*9.59*	2.83	2.39	2.3	1.51	1.04	•	1.31	1.36	1.98	1.49	1.77	2.17	2.45
	NVM root time	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	•	*1*	*1*	*1*	*1*	*1*	*1*	*1*
krob100	BP root %	**6.96**	**7.03**	**5.94**	**6.46**	**6.79**	**8.1**	**1.32**	**1.02**	**0.5**	**0.62**	**0.5**	**0.47**	**0.49**	**0.35**	**0.13**	**0.18**	**0.00**	**0.00**	•
	BP root time	3600	3600	3600	3600	3600	3600	3600	3600	1984	3600	3600	3600	3600	3600	2987	3600	3215	3474	•
	NVM root %	7.47	7.9	6.72	7.52	7.91	9.27	2.61	2.38	2.16	2.08	2	2.01	2.05	1.91	1.79	1.75	1.63	1.6	•
	NVM root time	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	•
krod100	BP root %	**3.33**	**8.35**	**1.73**	**1.11**	**0.82**	**0.55**	**0.26**	**0.06**	**0.02**	**0.07**	**0.09**	**0.15**	•	**0.1**	**0.07**	**0**	**0**	**0.15**	**0.21**
	BP root time	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	•	320	370	28	42	62	77
	NVM root %	4.11	9.62	2.53	2.27	2.09	2.06	1.79	1.62	1.58	1.63	1.65	1.71	•	1.66	1.63	1.74	1.88	2.19	2.47
	NVM root time	*1*	*1*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*1*	*2*	•	*1*	*2*	*2*	*2*	*2*	*2*
lin105	BP root %	12.05	13.64	9.67	5.31	5.24	**4.69**	**6.28**	**8.9**	**12.41**	**6.26**	**1.44**	**1.43**	**0.69**	**0.3**	**0.22**	**0.38**	**0.49**	**0.51**	•
	BP root time	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	475	179	88	•
	NVM root %	*10.29*	*12.66*	*9.04*	*5.12*	*5.22*	4.81	6.56	9.27	12.83	6.82	2.09	2.19	1.46	1.07	1.00	0.84	0.91	0.98	•
	NVM root time	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*2*	*2*	*2*	•
gr120	BP root %	**3.59**	4.98	**4.78**	**5.48**	**7.34**	**7.69**	**9.59**	**9.23**	**0.42**	**0.27**	**0.22**	**0.07**	**0.03**	•	**0.12**	**0.22**	**0.46**	**0.13**	**0.12**
	BP root time	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	•	3600	3600	3600	213	220
	NVM root %	3.6	*4.92*	4.93	5.68	7.59	7.99	9.92	9.61	0.86	0.74	0.69	0.54	0.5	•	0.59	0.69	0.93	0.96	1.11
	NVM root time	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	•	*2*	*2*	*2*	*2*	*2*
bier127	BP root %	5.36	**7.14**	**8.65**	**1.67**	**1.35**	**0.93**	**0.66**	**0.36**	**0.09**	**0.1**	•	**0.04**	**0.03**	**0.07**	**0**	**0.02**	**0.02**	**0.02**	**0.02**
	BP root time	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600	•	3600	3600	3600	86	147	184	129	127
	NVM root %	*4.52*	7.15	8.75	1.53	1.43	1.13	0.92	0.74	0.48	0.48	•	0.42	0.42	0.46	0.46	0.54	0.63	0.72	0.81
	NVM root time	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	•	*1*	*1*	*2*	*2*	*2*	*2*	*2*	*2*

Table 2.5: Performance of B&P and NVM at the respective root node for the medium TSPLIB instances and $p \leq 20$

Ins	Alg./p	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
st70	BP root %	**0.69**	**1.73**	**4.51**																	
	BP root time	27	31	51																	
	NVM root %	3.93	5.96	10.16																	
	NVM root time	*1*	*1*	*1*																	
eil76	BP root %	**0**	**0**	**0**	**0.69**	**0.84**															
	BP root time	37	34	31	40	65															
	NVM root %	4.23	4.81	5.7	7.22	8.98															
	NVM root time	*1*	*1*	*1*	*1*	*1*															
pr76	BP root %	**0**	**0**	**0.23**	**0.71**	**0.80**															
	BP root time	38	30	34	49	46															
	NVM root %	6.04	6.61	7.46	8.76	10.06															
	NVM root time	*1*	*1*	*1*	*1*	*1*															
gr96	BP root %	**0.06**	**0.25**	**0.48**	**0.66**	**0.71**	**0.54**	**0.44**	**1.01**	**1.99**	**3.31**	**6.03**	**5.90**								
	BP root time	88	60	53	67	77	87	63	114	66	84	80	143								
	NVM root %	2.04	2.44	2.94	3.39	3.7	3.86	4.28	5.4	6.94	8.96	12.52	13.71								
	NVM root time	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*								
rat99	BP root %	**0.06**	**0.09**	**0.10**	**0.10**	**0.10**	**0.14**	**0.14**	**0.17**	**0.17**	**0.24**	**0.20**	**0.50**	**1.41**							
	BP root time	93	91	83	86	80	123	128	81	104	74	106	99	156							
	NVM root %	4.31	4.77	5.22	5.67	6.11	6.63	7.13	7.7	8.27	8.97	9.58	10.66	12.43							
	NVM root time	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*							
rd100	BP root %	**0.13**	**0.11**	**0.03**	**0.01**	**0**	**0.10**	**0.21**	**0.24**	**0.20**	**0.19**	**0.42**	**0.61**	**4.22**							
	BP root time	97	83	81	87	51	77	82	72	87	101	75	79	69							
	NVM root %	2.73	2.8	2.8	2.86	2.94	3.35	3.83	4.23	4.57	4.96	5.63	6.32	10.52							
	NVM root time	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*							
kroa100	BP root %	**0.09**	**0.20**	**0.32**	**0.18**	**0.09**	**0.19**	**0.43**	**0.56**	**0.56**	**0.78**	**1.59**	**3.78**	**4.35**							
	BP root time	82	65	67	55	55	80	122	64	102	88	77	72	119							
	NVM root %	2.79	3.31	3.84	4.12	4.44	4.95	5.64	6.22	6.71	7.41	8.78	11.59	13.1							
	NVM root time	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*							
krob100	BP root %	**0.27**	**0.07**	**0.11**	**0**	**0.12**	**0.40**	**0.10**	**0**	**0**	**0.02**	**0.69**	**0.95**	**3.87**							
	BP root time	3600	958	3600	57	58	48	64	48	48	76	76	96	111							
	NVM root %	1.84	1.7	1.68	1.69	2.18	2.81	2.88	3.13	3.6	4.21	5.67	7.22	11.28							
	NVM root time	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*							
krod100	BP root %	**0.25**	**0.23**	**0.24**	**0.12**	**0.05**	**0.09**	**0.11**	**0.38**	**0.43**	**0.47**	**0.73**	**1.55**	**2.87**							
	BP root time	69	59	69	46	85	67	66	69	93	69	68	88	97							
	NVM root %	2.73	2.92	3.15	3.29	3.49	3.79	4.12	4.72	5.2	5.75	6.53	7.95	10.28							
	NVM root time	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*							
lin105	BP root %	**0.27**	**0.28**	**0.27**	**0.25**	**0.52**	**0.07**	**0**	**0.48**	**0.36**	**0.67**	**0.52**	**1.09**	**3.72**	**4.57**	**9.17**					
	BP root time	38	49	52	39	57	48	43	77	52	74	58	94	72	122	207					
	NVM root %	0.84	0.92	0.98	1.09	1.48	1.16	1.21	1.83	2.02	2.71	2.93	4	7.16	9.21	16.35					
	NVM root time	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*					
gr120	BP root %	**0**	**0**	**0.01**	**0.16**	**0**	**0**	**0.13**	**0.33**	**0.30**	**0.23**	**0.19**	**0.20**	**0.09**	**0**	**0**	**0.15**	**0.17**	**0.36**	**1.08**	**1.05**
	BP root time	110	97	156	137	134	125	161	168	175	175	167	155	118	87	108	126	159	166	156	323
	NVM root %	1.15	1.38	1.65	2.04	2.12	2.38	2.78	3.25	3.48	3.69	3.96	4.26	4.45	4.66	5.04	5.79	6.43	7.25	8.63	9.72
	NVM root time	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*
bier127	BP root %	**0.02**	**0.02**	**0**	**0**	**0.02**	**0**	**0**	**0**	**0**	**0.18**	**0.15**	**0.13**	**0.33**	**2.62**	**13.24**	**10.35**	**2.36**	**2.85**	**5.54**	**0.63**
	BP root time	152	149	95	104	173	101	125	101	119	174	111	145	129	191	132	173	146	192	211	210
	NVM root %	0.9	1.01	1.09	1.19	1.34	1.46	1.61	1.84	2.1	2.53	2.8	3.14	3.73	6.39	17.15	15	8.08	9.19	12.33	8.43
	NVM root time	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*

Table 2.6: Performance of B&P and NVM at the respective root node for the medium TSPLIB instances and $p > 20$

2.3.3 Detailed Performance Statistics

The results in Sections 2.3.1 and 2.3.2 suggest a correlation between the performance of B&P and the value of p as a function of its proximity to p_{2m}. This subsection discusses and explains the reasons for this phenomenon. Our discussion is based on Table 2.7 which shows the averages of several performance statistics of B&P for small and medium size TSPLIB instances. These averages were calculated using the data in Tables A.1-A.6, which, for completeness, are provided in §A.3 in the Appendix.

For every value of p as a function of its proximity to p_{2m} (as given in the first column of Table 2.7), Table 2.7 provides the average number of times B&P called the pricing problem (**#PS**), the average ratio of the number of occurrences of cases 2, 3, or 4 to **#PS** (columns entitled **C2%**, **C3%**, and **C4%**, respectively), the average number of columns generated by B&P (**#COL**), the average size of the set Q defined in Algorithm 3 (**|Q|**), and finally, the average number of iterations required before quitting Algorithm 3 (**#SP**). The statistics for case 1 were not reported since, by definition, it is invoked at most once. Moreover, Table 2.7 also contains a summary of important results from Tables 2.1-2.3 and Tables 2.4-2.6. Specifically, columns 9 to 14 in Table 2.7 provide the average B&P root gap percentage (**BPR%**), the average B&P root time (**BP RT**), the average NVM root gap percentage (**NVMR%**), the average NVM root time (**NVM RT**), the average B&P total time (**BP TT**), and the average NVM total time (**NVM TT**).

Table 2.7 explains the good performance of the B&P algorithm when p is greater than $p_{2m} + 3$ and its poor performance for p less than or equal to $p_{2m} + 3$. On one hand, when p is greater than $p_{2m} + 3$, Algorithm 3 (used to solve the NP-hard problem arising in case 4) is rarely invoked, whereas case 3 arises, on average, in

Table 2.7: Performance statistics for B&P and NVM as a function of the proximity of p to p_{2m}

p	C2%	C3%	C4%	#PS	#COL	$\|Q\|$	#SP	BP R%	BP RT	NVM R%	NVM RT	BP TT	NVM TT
$p_{2m}-18$	50	0	50	76	82	36	12	9.51	3600	*8.88*	*1*	3600	3600
$p_{2m}-17$	36	0	64	49	64	19	5	10.34	3600	*10.28*	*1*	3600	3600
$p_{2m}-16$	31	0	69	286	346	10	2	**7.81**	3600	7.88	*1*	3600	3600
$p_{2m}-15$	46	0	54	515	1008	19	3	**5.89**	3600	6.32	*1*	3600	3600
$p_{2m}-14$	50	0	50	621	1167	15	2	**6.02**	3600	6.57	*1*	3600	3600
$p_{2m}-13$	58	0	42	595	1224	42	9	**5.46**	3600	5.89	*1*	3600	3600
$p_{2m}-12$	47	0	53	305	629	57	16	**3.92**	3600	4.69	*1*	3600	*3443*
$p_{2m}-11$	58	0	42	201	429	97	21	**5.70**	3600	5.85	*1*	3600	*2633*
$p_{2m}-10$	56	0	44	409	807	33	8	5.62	3420	*5.35*	*1*	3600	*2029*
$p_{2m}-9$	57	0	43	618	1184	21	3	**5.64**	3600	5.96	*1*	3600	*2021*
$p_{2m}-8$	53	0	47	486	903	23	4	**4.39**	3600	4.85	*1*	3600	*1214*
$p_{2m}-7$	58	0	42	714	1386	16	2	**3.34**	3600	3.92	*1*	3600	*908*
$p_{2m}-6$	57	0	43	803	1416	15	2	**3.08**	3600	3.85	*1*	3600	*659*
$p_{2m}-5$	57	0	43	713	1196	15	2	**1.19**	2989	2.13	*0*	3024	*311*
$p_{2m}-4$	56	0	43	580	1022	20	4	**1.21**	2957	1.64	*0*	3187	*32*
$p_{2m}-3$	54	1	45	565	967	13	2	**0.42**	2514	1.59	*1*	2636	*9*
$p_{2m}-2$	57	2	41	506	875	15	2	**0.23**	1861	1.14	*1*	2615	*1*
$p_{2m}-1$	61	2	37	676	1051	14	2	**0.23**	1880	0.89	*1*	2109	*1*
p_{2m}	•	•	•	•	•	•	•	•	•	•	•	•	•
$p_{2m}+1$	72	16	12	685	1051	10	2	**0.26**	663	1.00	*1*	1739	*7*
$p_{2m}+2$	69	22	9	718	1091	10	2	**0.27**	441	1.27	*1*	1509	*168*
$p_{2m}+3$	71	28	1	626	928	4	1	**0.25**	547	1.57	*1*	1123	*770*
$p_{2m}+4$	65	34	1	528	807	5	1	**0.23**	47	1.94	*1*	**308**	784
$p_{2m}+5$	57	43	0	921	1232	4	1	**0.41**	53	2.63	*1*	**1080**	1605
$p_{2m}+6$	56	44	1	623	896	4	1	**0.86**	55	3.59	*1*	**1029**	2457
$p_{2m}+7$	52	48	0	573	824	3	0	**0.38**	56	2.99	*1*	**626**	3211
$p_{2m}+8$	52	48	0	362	625	1	0	**0.35**	63	2.99	*1*	**749**	3379
$p_{2m}+9$	52	48	0	566	850	2	0	**0.33**	64	3.54	*1*	**558**	3400
$p_{2m}+10$	42	58	0	453	684	0	0	**0.66**	69	4.61	*1*	**745**	3600
$p_{2m}+11$	53	47	0	299	574	0	0	**0.16**	70	3.26	*1*	**323**	3600
$p_{2m}+12$	47	53	0	464	721	0	0	**0.26**	76	3.82	*1*	**1187**	3600
$p_{2m}+13$	40	60	0	544	789	0	0	**0.79**	82	4.82	*1*	**1281**	3600
$p_{2m}+14$	34	66	0	521	748	0	0	**0.87**	85	5.13	*1*	**1393**	3600
$p_{2m}+15$	36	64	0	510	745	0	0	**1.20**	103	5.59	*1*	**1182**	3600
$p_{2m}+16$	39	61	0	377	614	0	0	**0.33**	90	4.86	*1*	**702**	3600
$p_{2m}+17$	35	65	0	397	637	0	0	**0.41**	83	5.45	*1*	**1469**	3600
$p_{2m}+18$	33	67	0	441	657	0	0	**0.64**	96	5.61	*1*	**2110**	3600
$p_{2m}+19$	34	66	0	373	573	0	0	**1.64**	78	7.16	*1*	**1974**	3600
$p_{2m}+20$	36	64	0	300	505	0	0	**1.32**	95	7.01	*1*	**1384**	3600
$p_{2m}+21$	29	71	0	386	592	0	0	**1.47**	100	7.18	*1*	**2063**	3600
$p_{2m}+22$	22	78	0	385	536	0	0	**1.93**	137	8.53	*1*	**2101**	3600
$p_{2m}+23$	30	70	0	337	587	0	0	**4.59**	126	10.70	*2*	**2736**	3600
$p_{2m}+24$	25	75	0	419	639	1	1	**3.87**	144	10.63	*2*	**2748**	3600
$p_{2m}+25$	28	72	0	366	599	0	0	**1.22**	181	8.92	*2*	**2800**	3600
$p_{2m}+26$	30	70	0	322	556	0	0	**1.53**	149	9.39	*2*	**1989**	3600
$p_{2m}+27$	19	81	0	388	558	0	0	**3.02**	155	11.50	*2*	**3331**	3600
$p_{2m}+28$	16	84	0	549	660	0	0	**1.02**	183	10.43	*2*	3600	3600

61% of the total calls to the pricing problem, and algorithm 2 (used to handle case 3) has polynomial time complexity. On the other hand, when p is less than or equal to $p_{2m}+3$, case 4 arises, on average, in 42% of the total calls to the pricing problem.

Even though B&P has a good performance when p is greater than $p_{2m}+3$, its performance starts deteriorating as p increases and more markedly as p approaches $\frac{|V|}{3}$. This is because the tightness of the root node's LP relaxation deteriorates as p approaches $\frac{|V|}{3}$ (see Table 2.7). Consequently, the size of the B&P tree increases substantially as p approaches $\frac{|V|}{3}$ (this is clearly shown in Tables A.1-A.6 in §A.3).

Next, we present an explanation for the dominance of occurrence of case 4 (compared to case 3) when $p < p_{2m}$ and vice versa when $p > p_{2m}$. To this end, given a graph G and the number of required cycles, p, let $F(p)$ be the optimal objective value of the HpMP on graph G. Figure 2.1 shows the plot of $F(p)$ versus p for graphs brazil58 and swiss42. This figure illustrates a nice property that we noticed when solving the HpMP on different graphs, that is, in general, $F(p)$ is monotonically increasing or decreasing on large intervals of p values. For example, the top graph in Figure 2.1 shows that for graph brazil58, $F(p)$ is monotonically decreasing on $1 \leq p \leq 12$ and is monotonically increasing on $12 \leq p \leq 19$, also the bottom graph in Figure 2.1 shows that for graph swiss42, $F(p)$ is monotonically decreasing for $1 \leq p \leq 5$ and is monotonically increasing on $7 \leq p \leq 14$. Thus, on one hand, when solving HpMP with $p < p_{2m}$ and $F(p)$ is monotonically decreasing on $[1, p]$, the constraint $\sum_{k \in K} x_k = p$ in Problem 2.1 is equivalent to $\sum_{k \in K} x_k \leq p$. Thus, its dual variable, μ_0, is less than or equal to zero (as Problem 2.1 is a minimization problem) and, consequently, case 3 will never occur. On the other hand, when solving HpMP with $p > p_{2m}$ and $F(p)$ is monotonically increasing on $[p, \lfloor \frac{|V|}{3} \rfloor]$, the constraint $\sum_{k \in K} x_k = p$ in Problem 2.1 is equivalent to $\sum_{k \in K} x_k \geq p$. Accordingly, $\mu_0 \geq 0$ and, consequently, case 4 will never occur. Evidently, not all graphs have this perfect monotonic be-

havior, however, this monotonicity is rarely violated. Accordingly, Table 2.7 shows some rare occurrences for cases 3 and 4 where $p < p_{2m}$ and $p > p_{2m}$, respectively.

The graph in Figure 2.1 also suggests an explanation for the good performance for NVM in the instances where $p_{2m} - 3 \leq p \leq p_{2m} + 3$. In particular, the objective value of these instances was relatively close to their respective 2-matching objective value. Thus, in general, since the core of the NVM is the 2-matching constraints, a smaller number of cuts to enforce the formation of exactly p cycles had to be added to these instances when compared to the instances where $p < p_{2m} - 3$ or $p > p_{2m} + 3$.

In summary, Table 2.7 shows that the average computational time for B&P is smaller when $p > p_{2m} + 3$, whereas the average time for NVM is smaller when $p \leq p_{2m} + 3$. It also shows that the values of BPR% and NVMR% are close when $p < p_{2m} - 3$, and they grow apart as p increases. Specifically, the average ratio of NVMR% to BPR% is 1.14 for instances with $p < p_{2m} - 3$, this ratio increases to 4.57 for instances with $p_{2m} - 3 \leq p \leq p_{2m} + 3$, and increases again to 7.73 for instances with $p > p_{2m} + 3$. Finally, judging by **#COL**, it seems that the B&P had trouble generating enough columns for very small p values (i.e., $p_{2m} - 18$ and $p_{2m} - 17$), this results in the poor values of the root gap percentage for these instances.

2.3.4 Larger Instances Results

The results in the previous subsections motivated us to test our B&P algorithm on larger instances for p values greater than $p_{2m} + 3$. We tested the algorithm on seven complete graphs with 150, 159, 200, 262, 299, and 318 nodes under varying values of p resulting in 273 instances. Table 2.8 presents the optimality gaps for different values of p where all these gaps were obtained at one hour time limit.

Table 2.8 shows the results of these 273 instances. We observed that, in 16% of these instances the optimality gap (OG) was less than 0.1%, in 37% of the instances

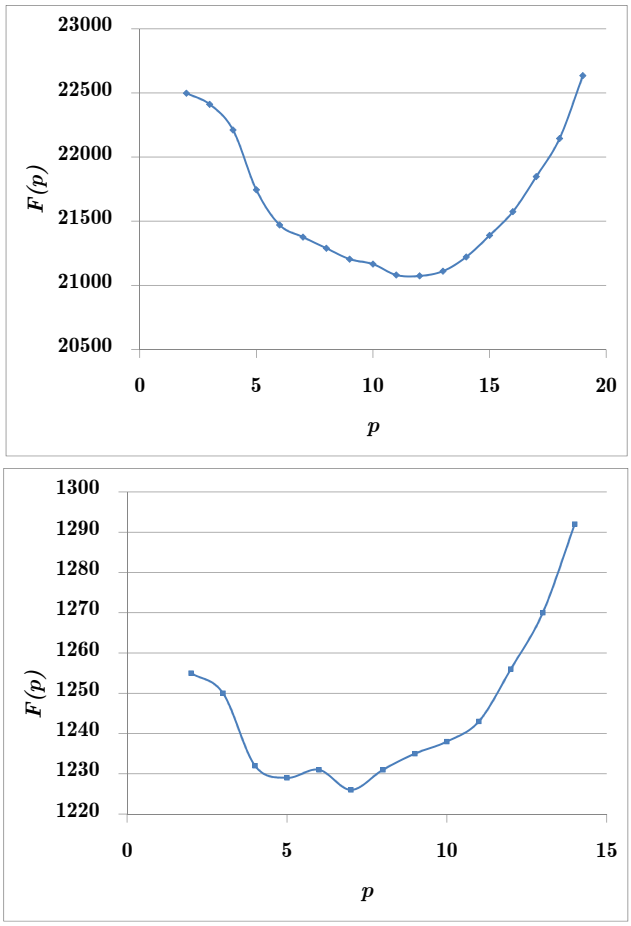


Figure 2.1: The optimal value of HpMP tends to be monotonically decreasing for $p < p_{2m}$ and monotonically increasing for $p > p_{2m}$. This behavior has implications in the performance of the algorithms to solve the HpMP. The top figure is for graph brazil58; whereas the bottom figure is for graph swiss42.

the OG was less than 0.5%, in 47% of the instances the OG was less than 1%, in 78% of the instances the OG was less than 5%, and in 87% of the instances the OG was less than 10%. Moreover, ten instances (2 instances from graph u159 and 8 instances from krob200) were solved to optimality and their computational times were 3381, 829, 3522, 1503, 2640, 2513, 3027, 2549, 3489, and 3378 seconds, respectively.

Table 2.8: Optimality gaps of B&P for large instances with $p > p_{2m} + 3$

kroa150		**u159**		**kroa200**		**krob200**		**gil262**		**pr299**				**lin318**	
p	gap%	p	gap%	p	gap%	p	gap%	p	gap%	p	gap%	p	gap%	p	gap%
25	2.86	22	0.12	35	0.17	35	0.57	35	0.27	30	0.26	81	1.76	63	0.61
26	0.50	23	0.05	36	0.23	36	3.63	36	5.93	31	2.97	82	0.16	64	0.27
27	0.36	24	0.06	37	1.74	37	1.66	37	16.59	32	15.20	83	0.35	65	0.25
28	0.33	25	0.08	38	0.29	38	0.93	38	1.71	33	16.37	84	1.71	66	0.32
29	0.25	26	0	39	0.88	39	1.24	39	0.05	34	28.09	85	4.07	67	0.66
30	0.39	27	0.01	40	0.95	40	0.40	40	4.50	35	1.44	86	2.41	68	1.08
31	0.18	28	0.18	41	0.12	41	0.04	41	0.57	36	1.37	87	0.07	69	0.84
32	0.09	29	0.15	42	0.16	42	0.08	42	0.36	37	1.87	88	1.38	70	0.55
33	0.06	30	0.01	43	0.07	43	0.01	43	14.78	38	3.87	89	0.94	71	0.57
34	0.11	31	0.04	44	0.03	44	0.14	44	0.71	39	1.84	90	2.78	72	0.46
35	0.05	32	0.10	45	0.08	45	0.02	45	12.03	40	9.07	91	5.46	73	0.25
36	0.11	33	0.19	46	0.11	46	0.08	46	0.20	41	2.15	92	1.38	74	0.16
37	0.14	34	0.03	47	0.48	47	0.04	47	3.90	42	6.05	93	5.05	75	0.31
38	1.44	35	0.09	48	1.21	48	0.09	48	0.77	43	2.94	94	3.14	76	0.03
39	0.18	36	0.31	49	0.08	49	0.04	49	2.39	44	12.34	95	8.99	77	0.34
40	1.29	37	0.13	50	0.28	50	0	50	1.62	45	1.70	96	6.52	78	1.21
41	1.73	38	0.04	51	1.79	51	0.04	51	1.96	46	3.43	97	4.77	79	4.50
42	1.17	39	0	52	0.67	52	0	52	1.75	47	2.89	98	5.38	80	3.10
43	0.17	40	0.04	53	0.48	53	0	53	0.68	48	2.86	99	7.33	81	1.83
44	1.06	41	0.27	54	2.06	54	0	54	1.78	49	4.41			82	4.29
45	1.26	42	0.44	55	0.93	55	0	55	1.05	50	2.53			83	0.31
46	2.36	43	0.26	56	1.36	56	0	56	0.26	51	4.65			84	3.60
47	4.89	44	0.63	57	2.46	57	0	57	1.56	52	5.03			85	1.79
48	2.18	45	2.65	58	8.57	58	0	58	0.09	53	3.11			86	12.30
49	7.35	46	0.43	59	6.19			59	0.04	54	59.33			87	5.48
50	8.71	47	1.00	60	23.93			60	0.11	55	11.97			88	11.21
		48	0.88					61	1.06	56	18.42			89	4.24
		49	2.28					62	0.27	57	2.10			90	14.01
		50	3.56					63	0.36	58	15.34			91	5.18
		51	3.4					64	0.48	59	19.18			92	13.92
		52	12.9					65	1.17	60	17.54			93	13.33
		53	8.47					66	0.94	61	7.07			94	11.45
								67	0.88	62	2.22			95	9.96
								68	0.88	63	3.05			96	8.82
								69	0.47	64	2.46			97	12.45
								70	1.23	65	33.71			98	11.68
								71	0.21	66	1.66			99	11.39
								72	0.12	67	2.93			100	6.08
								73	0.81	68	11.80			101	9.79
								74	1.22	69	4.16			102	13.71
								75	0.58	70	21.27			103	10.09
								76	0.38	71	1.81			104	14.31
								77	0.71	72	18.39			105	14.82
								78	0.33	73	1.22			106	28.52
								79	10.37	74	6.44				
								80	2.85	75	0.56				
								81	0.71	76	1.05				
								82	2.45	77	0.72				
								83	10.19	78	1.60				
								84	5.19	79	0.04				
								85	3.49	80	0.07				

2.4 Conclusions

In this chapter, we studied the Hamiltonian p-median problem (HpMP) which is a generalization of the Traveling Salesman Problem (TSP). We developed a B&P algorithm to solve the HpMP and compared our computational results to those of the NVM presented in [21].

This chapter includes several contributions on modeling, methodology, and computational aspects: 1) we modified the set partitioning formulation of HpMP proposed by [21]; 2) we developed a new efficient algorithm to find the shortest cycle in an undirected graph with arbitrary edge costs and no negative cycles; 3) we developed an algorithm to find the most negative cycle in an undirected graph with arbitrary edge costs; 4) computationally, the proposed algorithm for solving the HpMP outperformed the previously presented algorithms as it successfully solves instances up to 318 nodes, as opposed to other exact algorithms which solved instances up to 100 nodes; 5) we proved that for every value of p, the HpMP is NP-hard even when restricted to Euclidean graphs; and 6) we showed that the practical complexity of HpMP and the performance of the algorithms to solve it substantially depend on the relation between p and p_{2m} (the number of cycles in the 2-matching optimal solution), furthermore, we were able to explain the reason for the good performance of B&P when p is greater than $p_{2m} + 3$ and the reason for the good performance of NVM when p is between $p_{2m} - 3$ and $p_{2m} + 3$, inclusive.

The comparison of the computational results of our B&P algorithm and NVM from [21] presents an interesting strategy when solving the HpMP. We start by solving the minimum weight two-matching problem to find p_{2m}. If the value of the required cycles p is close to p_{2m}, (i.e., $p_{2m} - 3 \leq p \leq p_{2m} + 3$), using the first IP model in [21] is recommended. If $p > p_{2m} + 3$, it is much faster to solve HpMP

using the proposed B&P algorithm. Both algorithms perform poorly, especially in larger instances, whenever $p < p_{2m} - 3$ and further research is needed to solve these instances.

Finally, we note that [30] presented a variant of HpMP in which p is the upper limit on the number of required cycles. Here we define a new variant of HpMP in which the number of cycles is required to be at least p. Our B&P algorithm can be used to solve both of these HpMP variants. Interestingly, when solving the newly defined variant, our algorithms are guaranteed to solve the pricing problem in polynomial time. This is because by changing the equality constraint (2.1c) to a greater than or equal constraint, the dual variable of the modified constraint is always nonnegative, and therefore only cases 2 and 3 arise when solving the pricing problem.

3. THE SLIM BRANCH AND PRICE METHOD WITH AN APPLICATION TO THE HAMILTONIAN P-MEDIAN PROBLEM

3.1 Introduction

In this chapter, we propose a new exact optimization method, the Slim Branch and Price (SBP), that can be used to solve a large class of combinatorial optimization problems: those that are amenable to be solved by Branch and Price type algorithms and that have binary master problems whose solutions must have a pre-specified number of non-zero variables. This is a large and important class of problems as it includes several classical and fundamental problems such as capacitated vehicle routing problem and its variants [7], parallel machine scheduling [4] and its variants, capacitated p-median problem [32] and its variants, balanced disjoint rings problem [45], Hamiltonian p-median problem [33], k-clustering problem [22], and political districting problem [34].

The vast majority of the state-of-art successful exact algorithms for hard combinatorial optimization problems can be classified within either the branching framework or the cutting plane framework. On one hand, important examples of branching algorithms include Branch and Bound algorithms and Branch and Price (B&P) algorithms. On the other hand, pure cutting plane algorithms and Benders decomposition are important algorithms in the cutting plane framework. Some of the most effective methods combine ideas from these two frameworks as evident in Branch and Cut algorithms and Branch, Price and Cut. This chapter presents the Slim Branch and Price method which borrows ideas and improves upon these frameworks. The herein proposed SBP method can be interpreted within each of these two optimization frameworks as shown in the next two paragraphs.

From a branching framework perspective, the core idea of the SBP method is to improve the traditional branching scheme of B&P with the objective of exploring the problem's feasible region more efficiently and effectively. Specifically, the branching scheme in SBP involves splitting the feasible region of the current node into two parts: the first part (the exploration node) contains all the feasible solutions that are *distant* from the optimal solution of the LP relaxation of the current node; whereas the second part (the resolution node) contains all the feasible solutions that are *close* to the current node's optimal LP solution. The objective of creating the exploration node is to explore effectively the feasible region; specifically, it expedites the exploration of the entirety of the feasible region. Meanwhile, the objective of creating the resolution node is to efficiently find the best integer feasible solution in the neighborhood of the current node's optimal LP solution. It is important to remark that the resolution problem can be solved exactly extremely fast and without further branching or column generation. Consequently, the branching tree generated by SBP consists of one one main (exploration) branch and several (resolution) leaf nodes; thus creating the slim branching tree shown in Figure 3.1 from which the name of the method originated.

From a cutting plane perspective: given the optimal LP solution at an exploration node, the core idea of the SBP method is to add a linear inequality that is violated by this optimal LP solution and that significantly improves the optimal LP bound obtained from the next exploration node. However, in contrast to pure cutting plane algorithms, the cut added by SBP is invalid because, in addition to cutting the current optimal LP solution, it may excise a subset of the feasible region that could contain the optimal integer solution. Thus, adding a sequence of these aggressive but invalid cuts results in expedited improved LP bounds at exploration nodes but it also results in excising several feasible regions that may contain the optimal integer

solution(s) to the problem. Therefore, in order to guarantee the exactness of SBP, as explained in the branching framework earlier, one needs to solve the resolution problems where the feasible region of each resolution problem corresponds to each of these excised regions. Alternatively, one may generate and solve a single resolution problem whose feasible region is the union of all the aforementioned excised feasible regions. We show later that this merged resolution problem can be solved extremely fast without the need of further branching or column generation.

Finally, we remark that there is a tradeoff between the aggressiveness of the added cut (the improvement on the LB bounds in the exploration nodes) and the easiness of the resolution problem(s).

This chapter is organized as follows. Section 3.2 presents the Slim Branch and Price method; whereas Section 3.3 presents an implementation of SBP to solve the HpMP. Finally, Section 3.4 compares the computational results of SBP for HpMP versus those of the traditional B&P algorithm developed in Chapter 2.

3.2 Slim Branch & Price

This section describes the proposed Slim Branch & Price (SBP) method which is an improvement over traditional B&P in the case of binary master problems having fixed binary support (i.e., the summation of the variables in any feasible solution is fixed). This improvement is achieved by replacing the traditional branching scheme in B&P by a branching tree having one main branch and several leaves as shown in Figure 3.1. We refer to the nodes forming the main branch as ***exploration nodes*** and to the leaf nodes as ***resolution nodes***. The following two paragraphs give an overview of SBP while the ensuing subsections describe SBP in detail.

We call exploration procedure the process of solving the exploration nodes constituting the main branch in the branching tree. The feasible region of each exploration

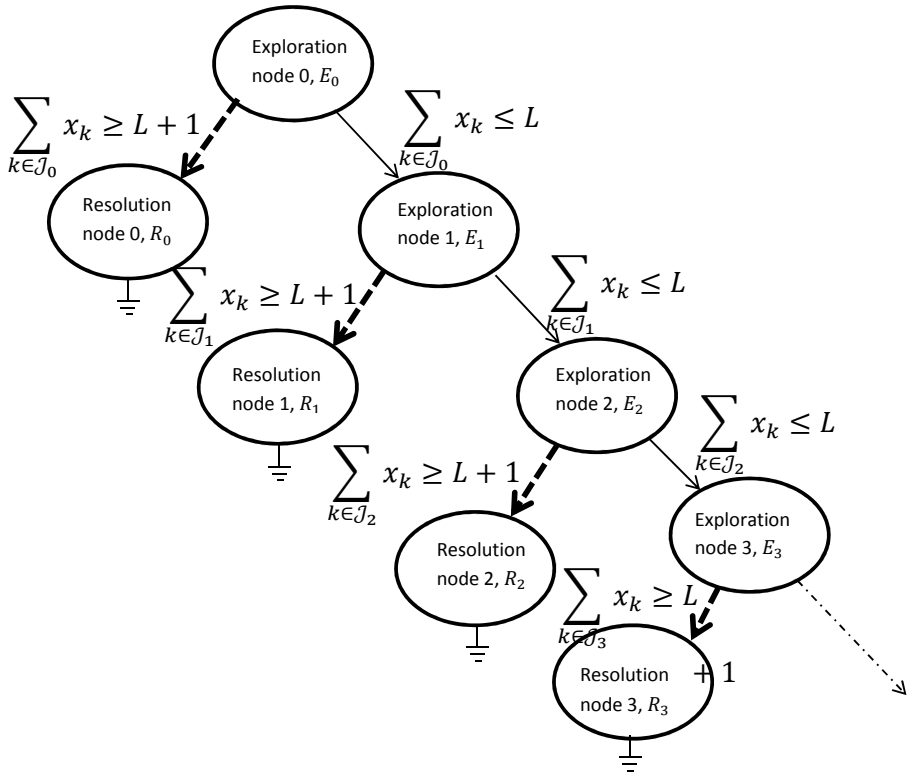


Figure 3.1: The Slim Branch & Price tree is composed of one main branch and several leaves. Here, $\mathscr{J}_t$ is the set of non-zero variables in the optimal LP solution of exploration node t. L is an algorithmic parameter balancing the aggressiveness of the exploration inequality and the easiness of the resolution problem(s). Section 3.2.2 explains the meaning of the dashed lines.

node is a subset of its parent's feasible region through the inclusion of exploration inequality. Adding a sequence of these inequalities results in an expedited increase in the value of the optimal solution of the LP relaxation of the restricted master problem (RMP) in the main branch. The availability of a good upper bound (provided by a heuristic or by solving the resolution nodes as discussed later) can greatly accelerate the termination of the exploration procedure.

We call resolution procedure the process of solving the resolution nodes that are represented as leaves in Figure 3.1. As discussed later, these problems, individually or collectively, are much easier to solve than the RMP and any optimization solver can quickly solve them exactly without the need of further branching or column generation. The importance of the resolution procedure is twofold: first, it guarantees the exactness of SBP; and second, the optimal solutions provided by solving the resolution nodes are valid upper bounds which can be used to accelerate the termination of the exploration procedure.

This remainder of this section gives a detailed presentation of SBP which is organized as follows. Section 3.2.1 presents the mathematical formulation of the RMP that is solvable using SBP. Section 3.2.2 presents the branching strategy in SBP. Sections 3.2.3 and 3.2.4 present the exploration and resolution procedures for the proposed SBP, respectively. Finally, Section 3.2.5 describes the different search strategies that can be used in SBP.

3.2.1 Master Problem in SBP

SBP can be applied to any optimization problem whose master problem has only binary variables and has fixed support (i.e., the number of columns with non-zero values in any feasible solution is fixed). Problem 3.1 presents a generic RMP which generalizes all the problems that can be solved using SBP. Without loss of generality,

we only consider minimization problems throughout our discussion in this section. Let T be the total number of exploration nodes solved until a stopping criterion is achieved, $\mathscr{C}_t$ be the set of columns generated prior and including exploration node $t = 0, \ldots, T$, and $\mathscr{C}_{-1}$ be the set of columns used to initialize column generation.

$$\textbf{(RMP)} \quad \text{minimize} \sum_{j \in \mathscr{C}_{-1}} c_j x_j \tag{3.1a}$$

$$\text{subject to} \sum_{j \in \mathscr{C}_{-1}} a_{ij} x_j = s_i, \qquad \forall i \in \mathscr{H}_1 \tag{3.1b}$$

$$\sum_{j \in \mathscr{C}_{-1}} b_{kj} x_j \geq t_k, \qquad \forall k \in \mathscr{H}_2 \tag{3.1c}$$

$$\sum_{j \in \mathscr{C}_{-1}} x_j = p \tag{3.1d}$$

$$x_j \in \{0, 1\}, \qquad j \in \mathscr{C}_{-1}. \tag{3.1e}$$

The distinctive characteristic of this RMP is the cardinality constraint (3.1d) which implies that the number of columns with non-zero values in any feasible solution is fixed and equal to p. Note that RMP contains as special cases the following two important classes of master problems: master problems that comprise, in addition to the cardinality constraint, only a block of set partitioning constraints or only a block of set covering constraints. These two special structures are the backbone of most master problems in the real life applications mentioned above.

In SBP, the root node is solved using column generation. Just like in B&P, in order to solve the LP relaxation of RMP at the root node, $\mathscr{C}_{-1}$ is used to initialize the column generation procedure. If available, RMP can be initialized using any heuristic integer solution; otherwise, artificial columns with high costs are used. The LP relaxation of RMP is then solved, and using the dual variables, the pricing problem is formulated and solved to generate a set of columns with negative reduced

costs. This process is repeated until no such column is available. Branching starts when the optimal solution at the root node is fractional; otherwise, an optimal integer solution is readily obtained. For further details, we refer the reader to [8].

3.2.2 Branching in Slim Branch and Price

The main difference between the proposed SBP and traditional B&P is in the adopted branching scheme. Let $\mathscr{J}_t, t = 0, \ldots, T$ be the set of columns with non-zero values in the optimal LP solution at exploration node t and L be an algorithmic parameter whose value is strictly less than p. After solving exploration node t (E_t), we then use the inequality

$$(\textbf{Exploration Inequality } \boldsymbol{t}) \quad \sum_{j \in \mathscr{J}_t} x_j \leq L \tag{3.2}$$

to force the feasible region of the next exploration node (E_{t+1}) to be distant from the current node's optimal LP solution. Complementarily, we use the inequality

$$(\textbf{Resolution Inequality } \boldsymbol{t}) \quad \sum_{j \in \mathscr{J}_t} x_j \geq L + 1 \tag{3.3}$$

to force the feasible region of the corresponding resolution node (R_t) to be close to the current node's optimal LP solution. This is shown in Figure 3.2. With a slight abuse of notation, we let R_t refer to both the t^{th} resolution node and its associated resolution problem; the same applies for E_t.

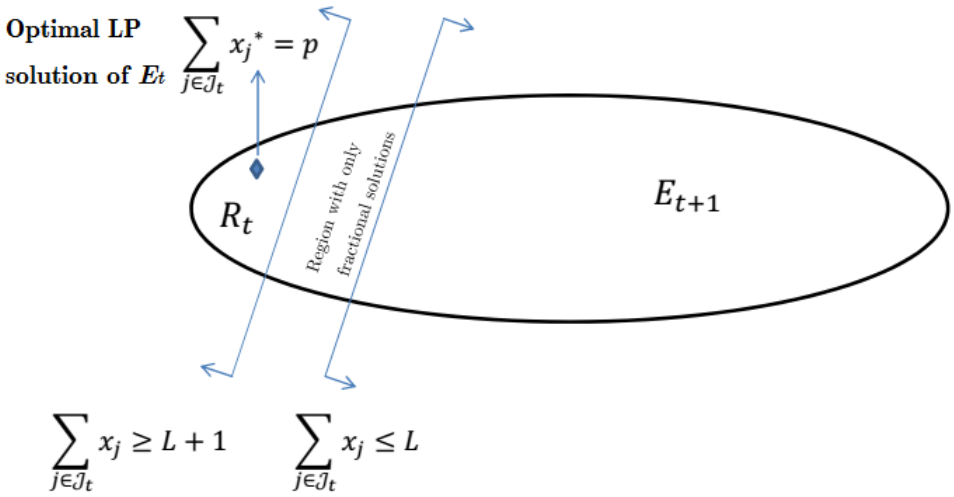


Figure 3.2: Splitting the feasible region between exploration and resolution problems.

Next, we explain the effect of adding the exploration inequality (3.2) on the optimal LP relaxation of an exploration node. Consider E_t and E_{t+1} on the main branch where the only difference between these nodes is that E_{t+1} includes the exploration inequality $\sum_{s \in \mathscr{J}_t} x_s \leq L$. This exploration inequality and the cardinality constraint (3.1d) imply that the summation of the values of the newly added variables (not in $\mathscr{J}_t$: variables with a zero value in the optimal LP solution of E_t or newly generated variables) when solving E_{t+1} has to be at least $p - L$. This implies that as L decreases (recall that $L < p$), the exploration inequality forces node E_{t+1} to explore a subset of the feasible region that is further away from the current LP solution at E_t (see Figure 3.2). Section 3.2.3 gives the details for the approach used to solve the exploration nodes.

Next, we present the reason for the easiness of the resolution problems $R_t, t = 0, \ldots, T$ when compared to solving the RMP, and consequently allowing us to solve them exactly as integer programs without the need of further branching or column

generation. The resolution inequality $\sum_{s \in \mathscr{J}_t} x_s \geq L + 1$ and the cardinality constraint (3.1d) enforce the optimal integer solution of R_t to include at most $p - (L + 1)$ columns different from the columns already in $\mathscr{J}_t$. For example, if $L = p - 1$, only the columns in $\mathscr{J}_t$ can be non-zero in the optimal integer solution of R_t, and thus, no more columns need to be generated. While if $L = p - 2$, at most one column outside of $\mathscr{J}_t$ will be needed to get the p pre-specified columns in the optimal integer solution of R_t. Thus, most of the columns forming an optimal solution are already known, therefore R_t is much easier to solve compared to RMP. Clearly, the difficulty of R_t increases as L decreases.

It is important to clarify that R_t does not include any of the exploration inequalities from its predecessors. Specifically, R_t includes only the t^{th} resolution inequality (3.3); whereas all the preceding t exploration inequalities are discarded in formulating R_t. Figure 3.1 depicts this fact by using dashed lines to connect the resolution node to its parent. Discarding these t exploration inequalities is a secondary reason for the easiness of the resolution problem, and most importantly allows us to combine all the resolution nodes and solving them as a single integer program. Section 3.2.4 presents some guidelines for formulating and solving the resolution problems without the need of using the pricing problem to generate the very few columns that are needed in addition to the columns in $\mathscr{J}_t$.

Finally, we want to remark that the value of L plays an important role in SBP as it balances the aggressiveness of the exploration inequality and the easiness of the resolution problem as depicted in Figure 3.2. The closer the value of L to p, the easier the resolution problems (as fewer extra columns need to be formed) but the slower the increase in the objective value of the LP relaxation at the exploration nodes.

The most important aspect to consider when selecting the value of L is to obtain

a resolution problem that can be solved quickly. To this end, for each value of L, a model or algorithm is developed to obtain at most $p-(L+1)$ new columns in addition to using at least $L+1$ of the columns in $\mathscr{J}_t$. As mentioned earlier, the cases with $L=p-1$ and $L=p-2$ lead to very easy resolution problems. In general, we recommend setting the value of L close (but not extremely close) to p. Section 3.3.3 explains examples of resolution problems formulations for HpMP for two different values of L: $L=p-2$, and $L=p-3$.

3.2.3 Solving an Exploration Problem

This section explains how to formulate and solve each of the exploration problems of any optimization problem that is amenable to be solved with SBP. The only difference between the root-node problem, E_0, and the subsequent exploration problem, $E_t, \forall t=1,\ldots,T$, is the inclusion of the exploration inequalities. We give below the generic formulation of the RMP of exploration node $t+1$ followed by the derivation of the associated pricing problem.

$$
\begin{aligned}
(E_{t+1}) \quad \text{minimize} \quad & \sum_{j\in\mathscr{C}_t} c_j x_j && && \text{(3.4a)}\\
\text{subject to} \quad & \sum_{j\in\mathscr{C}_t} a_{ij} x_j = s_i, && \forall i\in\mathscr{H}_1 && \text{(3.4b)}\\
& \sum_{j\in\mathscr{C}_t} b_{kj} x_j \geq t_k, && \forall k\in\mathscr{H}_2 && \text{(3.4c)}\\
& \sum_{j\in\mathscr{C}_t} x_j = p && && \text{(3.4d)}\\
& \sum_{j\in\mathscr{J}_k} x_j \leq L && \forall k\in\{0,\ldots,t\} && \text{(3.4e)}\\
& x_j\in\{0,1\} && \forall j\in\mathscr{C}_t && \text{(3.4f)}
\end{aligned}
$$

To derive the pricing problem associated with Problem 3.4, let $\alpha_i, \forall i\in\mathscr{H}_1$ and

$\beta_k, \forall k \in \mathscr{H}_2$ be the dual variables associated with constraints (3.4b) and (3.4c), respectively, and let μ_0 be the dual variable associated with constraint (3.4d). Note that the constraints (3.4e) are the exploration inequalities which only include the columns generated previously (during E_0 to E_t); consequently, the dual variables of these constraints are not needed in the pricing problem. With this in mind, the objective function of the pricing problem is

$$\min_{a^i, i\in\mathscr{H}_1; b^k, k\in\mathscr{H}_2} Z = \bar{c} - \left(\sum_{i\in\mathscr{H}_1} a^i \alpha_i + \sum_{k\in\mathscr{H}_2} b^k \beta_k + \mu_0 \right) \tag{3.5}$$

where a^i is related to the i^{th} constraint in $\mathscr{H}_1$, b^k is related to the k^{th} constraint in $\mathscr{H}_2$, and $\bar{c}$ is the cost of the column to be generated. As in traditional B&P, the optimal values of a^i and b^k will be the coefficients of the generated column in constraints (3.4b) and (3.4c), respectively; whereas the value of $\bar{c}$ will be the coefficient of the generated column in the objective function (3.4a).

The constraints in the pricing problem depend on the specific problem at hand. In general, constraints are needed to enforce the generated columns to be consistent with the underlying structure of the problem (e.g. columns must form cycles in HpMP, paths in parallel machine scheduling, cycles starting and ending at the depot in vehicle routing problem). Moreover, due to constraints (3.4e), the pricing problem must also include constraints to prevent the regeneration of the columns in $\mathscr{J}_0 \cup \ldots \cup \mathscr{J}_t$ (but we do not need analogous constraints for the already generated columns not in $\mathscr{J}_0 \cup \ldots \cup \mathscr{J}_t$). Note that the structure of these regeneration-prevention constraints also depends on the specifics of the problem studied (e.g. regeneration of cycles can be prevented using a form of subtour elimination constraints).

We end this section with a remark about the effect of the parameter L on solving the LP relaxation of exploration node E_{t+1}. As L decreases (moves away from p), the

LP relaxation of E_t provides better lower bounds because the exploration inequalities are more aggressive. However, as L decreases, the difficulty of E_{t+1} increases slightly because increasing the aggressiveness of the exploration inequalities (3.4e) decreases the amount by which the already generated columns in $\mathscr{J}_0 \cup \ldots \cup \mathscr{J}_t$ can contribute to the fixed support (3.4d), and consequently more columns may need to be generated to obtain the optimal LP solution to problem E_{t+1}.

3.2.4 Solving a Resolution Problem

This section provides some guidelines for formulating and solving the resolution problems of any optimization problem that is amenable to be solved with SBP. Solving each resolution problem R_t, left child in the branching, entails searching for an optimal integer solution in the neighborhood of the current LP solution of its corresponding exploration node E_t. Here, it is important to note two distinctions between SBP and B&P: 1) R_t is solved exactly without further branching; and 2) in order to facilitate solving it, R_t does not include any of the preceding t exploration inequalities whereas each node in B&P includes the branching constraints added to all its predecessors. Moreover, compared to traditional B&P, discarding these inequalities not only makes the resolution problem easier to solve to optimality but it also may improve the quality of the upper bound (integer feasible solution) obtained.

Formulating the resolution problem heavily depends on the studied optimization problem, and therefore it is not possible to give an explicit generic formulation (except for the case when $L = p - 1$, whose formulation is given below). However, regardless of the studied problem, the formulation of the resolution problem must adhere to the following guidelines. First, the formulation of R_t should include a constraint that enforces the feasible solutions of R_t to contain at least $L + 1$ columns from $\mathscr{J}_t$. Second, the formulation of R_t should include one variable for each column in $\mathscr{J}_t$

and all the natural space variables for the studied optimization problem (e.g. edges in HpMP and vehicle routing problem). These natural space variables are used to form the remaining (at most) $p-(L+1)$ columns (e.g. cycles in HpMP and vehicle routing problem, paths in parallel machine scheduling). The objective is that this formulation can then be solved exactly without any calls to the pricing oracle using any off-the-shelf solver.

Even though we are not giving an explicit generic formulation for the resolution problem, we believe that it is relatively straightforward to develop the resolution problem's formulation for any given problem. To this end, one approach is to adapt the formulation and solution strategies for HpMP (given in Section 3.3.3) to the problem of interest.

Clearly, when $L = p - 1$, the formulation of R_t does not require natural space variables because only the columns in $\mathscr{J}_t$ are used in any feasible solution of R_t. Since the cardinality of $\mathscr{J}_t$ is, in general, small, this problem is usually very easy to solve (took less than one second in all of our computational experiments). The generic formulation for R_t when $L = p - 1$ is:

$$\text{minimize} \sum_{j \in \mathscr{J}_t} c_j x_j \tag{3.6a}$$

$$\text{subject to} \sum_{j \in \mathscr{J}_t} a_{ij} x_j = s_i, \qquad \forall i \in \mathscr{H}_1 \tag{3.6b}$$

$$\sum_{j \in \mathscr{J}_t} b_{kj} x_j \geq t_k, \qquad \forall k \in \mathscr{H}_2 \tag{3.6c}$$

$$\sum_{j \in \mathscr{J}_t} x_j = p \tag{3.6d}$$

$$x_j \in \{0, 1\} \qquad \forall j \in J_t \tag{3.6e}$$

3.2.4.1 Solving Several Resolution Problems at Once

Here, we explain how to combine resolution problems R_a to R_b and solve them collectively for any a, b such that $0 \leq a < b \leq T$. This property will be exploited in the search strategies given in Section 3.2.5. Since we are discarding the exploration inequalities in the resolution problems' formulation, we can formulate a *merged resolution problem* whose feasible region is a superset of the union of all the feasible regions of individual resolution problems. The formulation and solution strategies of the merged resolution problem are identical to those of the individual resolution problems; the only difference is that the merged formulation includes all of the columns in $\mathscr{J}_a \cup \ldots \cup \mathscr{J}_b$ whereas the individual resolution problems R_a to R_b use only columns $\mathscr{J}_a, \ldots, \mathscr{J}_b$, respectively. Consequently, the merged resolution problem can also be solved exactly using any off-the-shelf solver without calls to the pricing oracle. Moreover, since the merged resolution problem is a relaxation of each of the individual resolution problems, by solving it, one may even obtain a better feasible integer solution to the studied optimization problem (i.e., stronger upper bound) than those obtained by solving each individual resolution problem.

3.2.5 SBP Search Strategies

Based on the order in which the exploration and the resolution nodes are solved as well as whether the resolution nodes are solved individually or collectively, we define three general search strategies for SBP.

3.2.5.1 First Strategy

The first strategy alternates between solving an exploration node E_t and solving a resolution node R_t. The bold numbers beside the nodes in Figure 3.3 show the order in which the nodes are solved. Since the resolution nodes are solved to

optimality, the lower bound and the upper bound (the incumbent solution) may be updated after solving each exploration and resolution nodes, respectively. Therefore, the termination conditions for this search strategy are: (1) The LP solution to an exploration node has a higher objective value than the incumbent solution (in which case the incumbent solution is optimal). (2) If an exploration node is infeasible and there is an incumbent solution, then this solution is optimal. (3) If an exploration node is infeasible and there is no incumbent solution, then the problem instance is infeasible. Found at the end of this section, Lemma 3.2.1 proves that the number of exploration nodes is always finite.

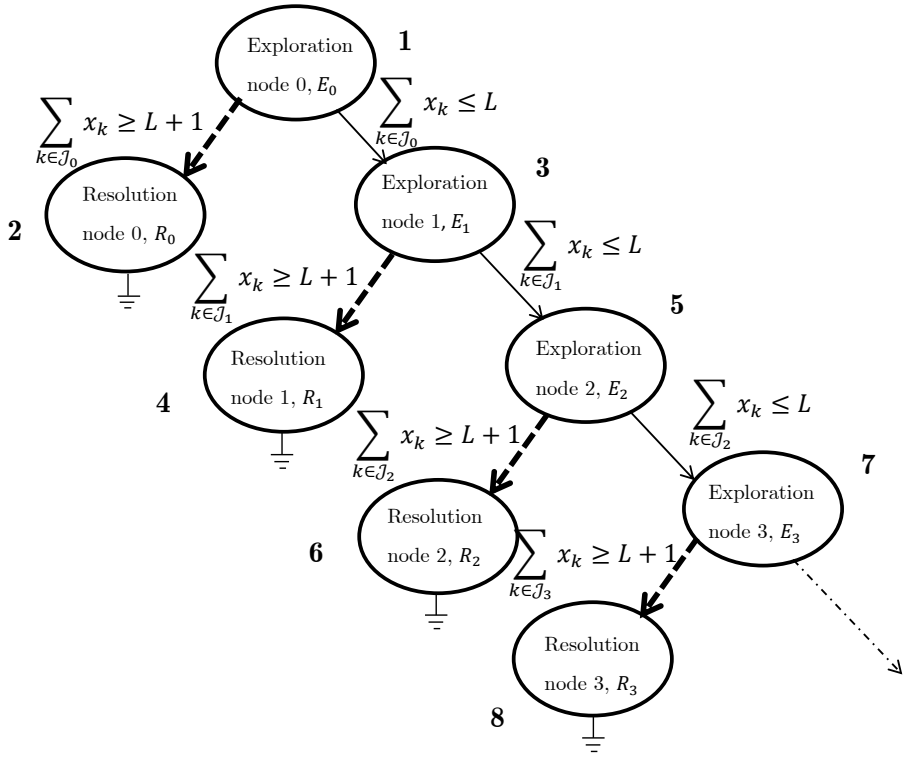


Figure 3.3: The first strategy alternates between solving an exploration node E_t and a resolution node R_t.

3.2.5.2 Second Strategy

Given an integer parameter $u \geq 2$, the second strategy alternates between solving u exploration nodes and solving u resolution nodes as a single merged resolution problem as explained in the end of Section 3.2.4. The bold numbers beside the nodes in Figure 3.4 show the order in which the nodes are solved when $u = 2$. Like in the first strategy, the upper bound may be updated every time a merged resolution problem is solved. However, the lower bound cannot be updated after solving each exploration problem; it can only be updated (1) after solving a merged resolution problem (in which case the lower bound is provided by the latest-solved exploration problem); and (2) after solving E_{ku} for $k = 0, \ldots, \lfloor \frac{T}{u} \rfloor$. The termination conditions of this strategy are the same as those of the first strategy.

3.2.5.3 Third Strategy

The objective in this strategy is to accelerate the termination of the exploration procedure (and consequently of the whole algorithm) by solving only a reduced version of the resolution problems; solving such reduced problems provides, in an expedited manner, high quality (improving) upper bounds. This search strategy is motivated by these two observations: 1) although the individual resolution problems can be solved very fast, solving them repeatedly adds up to a non trivial amount of time; whereas 2) the individual resolution problems in the case when $L = p - 1$ can be solved significantly faster than the resolution problems when $L < p - 1$. Therefore, we split R_t into two disjoint problems: the first easier problem, R'_t, uses only the columns in $\mathscr{J}_t$ to find an optimal p columns (this corresponds to a resolution problem when $L = p - 1$); while the second problem, R''_t, uses at most $p - 1$ columns from $\mathscr{J}_t$ when selecting an optimal p columns. The specific constraints included in R'_t and R''_t are given in Figure 3.5. Note that strictly speaking, when splitting the

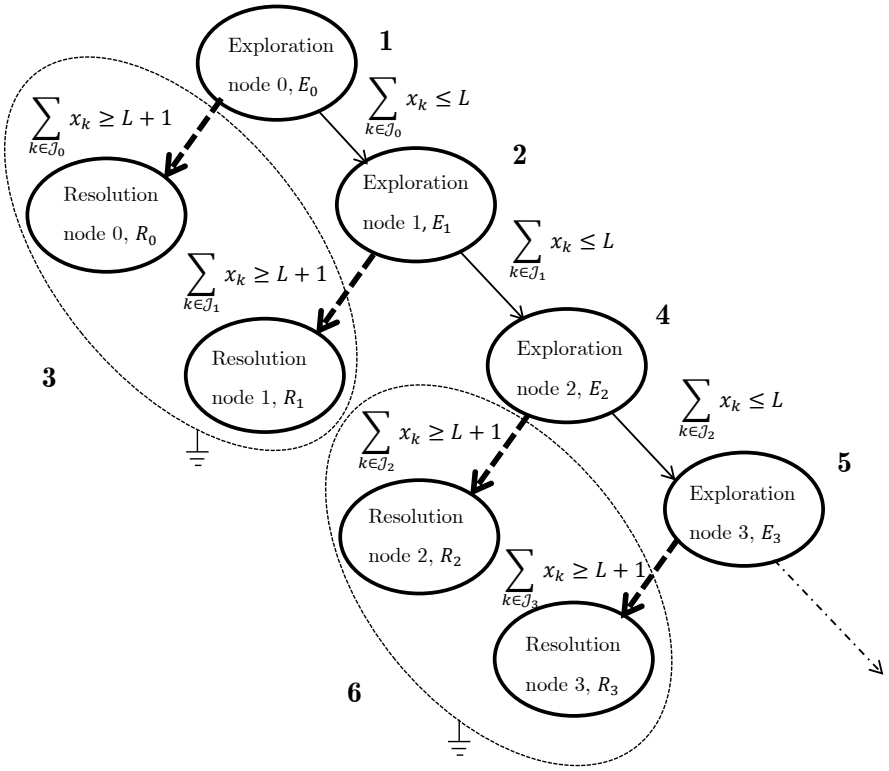


Figure 3.4: Second strategy (shown for $u = 2$). The second strategy alternates between solving u exploration nodes and u resolution nodes as a single merged resolution problem.

resolution problem R_t into the two subproblems, the problem R_t' should contain a constraint of the form $\sum_{j\in\mathscr{J}_t} x_j = p$ as opposed to the constraint given in Figure 3.5 ($\sum_{j\in\mathscr{J}_0\cup\ldots\cup\mathscr{J}_t} x_j = p$). With this modification, one may get stronger upper bounds when solving to optimality R_t' while incurring only an insignificant overhead.

Figure 3.5 shows the order in which this strategy traverses the branching tree. Specifically, this strategy starts by alternating between solving an exploration node E_t and solving R_t' until either the LP solution of an exploration node is greater than or equal to an incumbent solution or an exploration node is infeasible. Then, the strategy concludes by solving a single merged resolution problem which combines

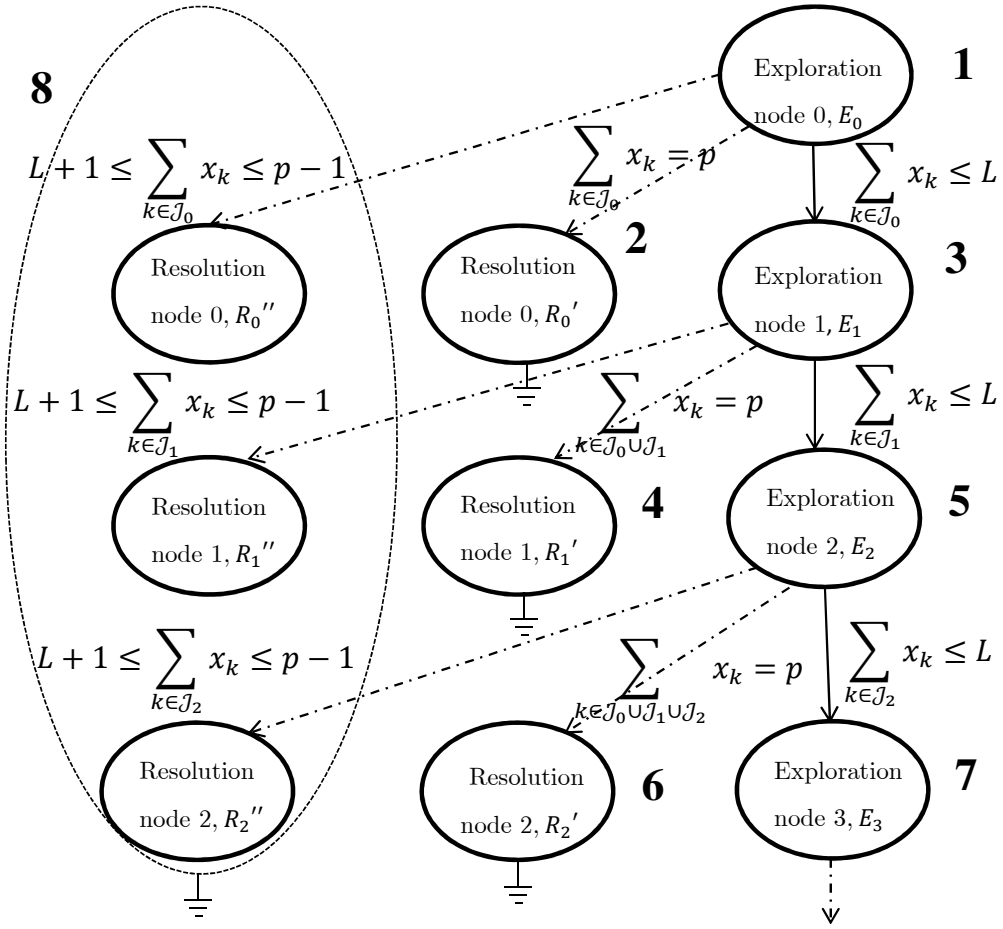


Figure 3.5: The third strategy alternates between solving an exploration node E_t and the *simpler* resolution problem $R_t^{'}$. The strategy concludes by solving the *harder* resolution problems $R_t^{''}$ for $t = 1, \ldots, T$ as a single merged resolution problem.

all the individual $R_t^{''}$ problems to possibly obtain a better upper bound (than that obtained by solving the easier resolution problems $R_t^{'}$) to declare as an optimal solution; otherwise, the best upper bound provided by solving the individual $R_t^{'}$ problems is an optimal solution.

Like in the first two strategies, the upper bound (and the corresponding incumbent solution) may be updated every time a (reduced) resolution problem $R_.^{'}$ is solved. Unlike the first two strategies, the LP solutions at the exploration nodes do not give

valid lower bounds because the nodes R_t'' are open (and, since we will solve them to optimality at once, we do not even calculate their LP solutions when they are first created). Therefore, before the algorithm's termination, the only valid lower bound is provided by the LP solution of the root node, E_0.

We end this section by presenting a lemma that proves that, regardless of the upper bound quality or availability, the exploration procedure terminates in a finite number of steps (i.e., the number of exploration nodes, and therefore, the number of resolution nodes, is finite).

Theorem 3.2.1. *The number of exploration nodes is finite.*

Proof. The proof uses the following three facts: 1) the number of columns in the master problem is finite; 2) the pricing problem explicitly prevents regeneration of the already generated columns; and 3) the exploration inequalities restrict $\mathscr{J}_{t+1}$ for $0 \leq t \leq T-1$ from containing more than L columns from any of the subsets $\mathscr{J}_0, \ldots, \mathscr{J}_t$. From (1), it follows that the cardinality of $\mathscr{J}_t$ is finite for every $0 \leq t \leq T$, while (2), (3), and the fixed support (equation (3.4d)) imply that all of the sets $\mathscr{J}_t$ for $0 \leq t \leq T$ are distinct from each other. Therefore, since the subsets $\mathscr{J}_t$ for $0 \leq t \leq T$ have finite cardinality and are distinct, there is only a finite number of such subsets. Consequently, since there is a one-to-one correspondence between exploration nodes and subsets $\mathscr{J}_t$ for $0 \leq t \leq T$, the result then follows. □

3.3 Solving Hamiltonian p-median Problem Using Slim Branch & Price

This section explains how the third search strategy of SBP is used to solve the HpMP. In Section 3.3.1, we present the RMP for HpMP and then briefly explain how column generation can be used to solve the root node. In Section 3.3.2, we explain how the exploration procedure in SBP is implemented to solve the main branch

nodes for HpMP. Finally, the procedure to solve the combined resolution problem(s) for HpMP is presented in Section 3.3.3 .

3.3.1 Solving the Root Node's Linear Relaxation

Recall that a detailed discussion of this column generation algorithm is presented in Chapter 2. The RMP for HpMP can be stated as follows:

$$\text{minimize} \sum_{k \in \mathscr{C}_{-1}} c_k x_k \tag{3.7a}$$

$$\text{subject to} \sum_{k \in \mathscr{C}_{-1}} a_{ik} x_k = 1 \qquad \forall i \in V \tag{3.7b}$$

$$\sum_{k \in \mathscr{C}_{-1}} x_k = p \tag{3.7c}$$

$$x_k \in \{0, 1\} \qquad \forall k \in \mathscr{C}_{-1} \tag{3.7d}$$

In column generation, our target is to find columns with negative reduced cost to add to the RMP. By defining another graph $G' = (V, E)$ with edge weights $d_{ij} - \frac{\mu_i}{2} - \frac{\mu_j}{2}$ where $\mu_i, \forall i = 1, \ldots, |V|$ are the dual variables associated to constraints (3.7b), we already established (in Chapter 2) that finding a column with a negative reduced cost in G is equivalent to finding a cycle in G' with total weight that is less than μ_0 where μ_0 is the dual variable for constraint (3.7c). Section 2.2.3 presents a detailed explanation of how the pricing problem was solved.

3.3.2 Exploration Procedure for HpMP

Exploration procedure entails solving the exploration nodes which constitute the main branch in SBP tree. We start by presenting the mathematical formulation for any exploration problem E_t for HpMP, followed by a discussion of formulating and solving the pricing problem.

The procedure starts by solving the first exploration node (i.e., E_0), which is the root node, as explained in Section 3.3.1. The exploration problem $E_{t+1}, t \geq 0$ can then be formulated as follows:

$$
\begin{aligned}
(E_{t+1}) \quad \text{minimize} \quad & \sum_{k \in \mathscr{C}_t} c_k x_k && && (3.8a)\\
\text{subject to} \quad & \sum_{k \in \mathscr{C}_t} a_{ik} x_k = 1, && \forall i \in V && (3.8b)\\
& \sum_{k \in \mathscr{C}_t} x_k = p && && (3.8c)\\
& \sum_{k \in \mathscr{J}_j} x_k \leq L && \forall j \in \{0, \ldots, t\} && (3.8d)\\
& x_k \in \{0, 1\} && \forall k \in \mathscr{C}_t && (3.8e)
\end{aligned}
$$

Let $\mu_i, \forall i \in V$ be the dual variables associated with constraints (3.8b), and let μ_0 be the dual variable associated with constraint (3.8c). As mentioned in Chapter 2, solving the pricing problem entails finding a cycle in G' whose weight is less than μ_0 or conclude that none exists. The IP used to solve the pricing problem can be formulated as:

$$
\begin{aligned}
\textbf{(IP-Pricing Problem)} \quad \text{minimize} \quad & Z = \sum_{(i,j) \in E} (d_{ij} - \frac{\mu_i}{2} - \frac{\mu_j}{2}) y_{ij} && && (3.9a)\\
\text{subject to} \quad & \sum_{(i,j) \in \delta(i)} y_{ij} = 2 z_i, && \forall i \in V && (3.9b)\\
& \sum_{(i,j) \in E} y_{ij} \geq 3 && && (3.9c)\\
& \sum_{\forall (i,j) \in q} y_{ij} \leq |q| - 1, && \forall q \in \mathscr{J} && (3.9d)\\
& y_{ij} \in \{0, 1\} && \forall (i,j) \in E && (3.9e)
\end{aligned}
$$

$$z_i \in \{0,1\} \qquad \forall i \in V \qquad (3.9f)$$

The variable y_{ij} is binary with value of 1 if the edge (i,j) is in the optimal solution; and zero otherwise. The constraints (3.9b) and (3.9c) ensure that the variables y_{ij} in any feasible solution form at least one cycle (recall that $\delta(i)$ is the set of incident edges to node $i \in V$). On the other hand, the constraints (3.9d) prevent regenerating the already generated columns in the set $\mathscr{J} = \mathscr{J}_0 \cup \ldots \cup \mathscr{J}_t$.

Algorithm 4, given below, uses Problem 3.9 to find a cycle in G' whose weight is less than μ_0 or conclude that none exists:

Algorithm 4 Solving the Pricing Problem

Input: $G' = (V, E)$, a weight $d_{ij} - \frac{\mu_i}{2} - \frac{\mu_j}{2}$ for all edges $(i,j) \in E$, and a number, μ_0.
Output: A cycle or set of cycles such that the weight of each cycle is less than μ_0 or conclude that no such cycle exists.
Set Stop=0.
repeat
    Solve Problem 3.9. Let the cycles in the optimal solution be $C_1, \ldots, C_s$ with weights $W_1, \ldots, W_s$. Note that $Z^* = \sum_{i=1}^{s} W_i$.
    if at least one $W_i, \forall i = 1, \ldots, s$ is less than μ_0 **then**
        return cycle(s) with W_i less than μ_0. Set Stop=1.
    else if $Z^* \geq \mu_0$ **then**
        return no cycle has W_i less than μ_0. Set Stop=1.
    else
        add $C_1, \ldots, C_s$ to $\mathscr{J}$.
    end if
until Stop=1

3.3.3 Resolution Procedure for HpMP

Preliminary experimental results show that when solving the HpMP, SBP has the best performance when $L = p - 2$ and $L = p - 3$. Thus, following this observation, our discussion will be divided into two main sections.

3.3.3.1 Resolution Procedure for $L = p - 2$

In this case, the resolution node t will contain the constraint $\sum_{k \in \mathscr{J}_t} x_k \geq L + 1 = p - 1$. Hence, we divide our discussion into two main parts. In the first part, we explain how the resolution problem, termed R_t^{p*}, which includes the constraint $\sum_{k \in \mathscr{J}_t} x_k = p$ is formulated and solved; whereas in the second part, we present the resolution problem, termed $R_t^{p-1\dagger}$, which includes the constraint $\sum_{k \in \mathscr{J}_t} x_k = p - 1$.

After solving the exploration node t, the corresponding resolution problem, R_t^p, can be formulated as follows:

$$(R_t^p) \quad \text{minimize} \sum_{k \in \mathscr{J}} c_k x_k \tag{3.10a}$$

$$\text{subject to} \sum_{k \in \mathscr{J}} a_{ik} x_k = 1, \qquad \forall i \in V \tag{3.10b}$$

$$\sum_{k \in \mathscr{J}} x_k = p \tag{3.10c}$$

$$x_k \in \{0, 1\} \qquad \forall k \in \mathscr{J} \tag{3.10d}$$

Thus, R_t^p requires finding a local optimal integer solution of HpMP by using as variables just the columns in $\mathscr{J} = \mathscr{J}_0 \cup \ldots \cup \mathscr{J}_t$. If a feasible solution for R_t^p exists, then it is clearly an upper bound for HpMP. Since the number of columns in $\mathscr{J}$ is

*This is the same as $R_t^{'}$ presented earlier but the notation is slightly changed to facilitate exposition specific to HpMP below.

†Analogous to $R_t^{''}$

usually small, solving R_t^p is easy and can readily be solved using any integer solver. For this reason and because no starting feasible solution is available, we solve the corresponding R_t^p after solving each exploration node E^t (as mentioned in the third search strategy in Section 3.2.5).

Next, we present the formulation of R_t^{p-1} below and then present an algorithm to solve it.

$$(R_t^{p-1}) \quad \text{minimize} \sum_{\forall e \in E} d_e y_e + \sum_{k \in \mathscr{J}_t} c_k x_k \tag{3.11a}$$

$$\text{subject to} \sum_{\forall e \in \delta(i)} y_e + \sum_{k \in \mathscr{J}_t} 2a_{ik} x_k = 2, \qquad \forall i \in V \tag{3.11b}$$

$$\sum_{k \in \mathscr{J}_t} x_k = p - 1 \tag{3.11c}$$

$$\sum_{e \in E} y_e \geq 3 \tag{3.11d}$$

$$x_k \in \{0, 1\} \qquad \forall k \in \mathscr{J}_t \tag{3.11e}$$

$$y_e \in \{0, 1\} \qquad \forall e \in E \tag{3.11f}$$

We restrict the feasible region of R_t^{p-1} to select exactly $p-1$ columns (cycles) from $\mathscr{J}_t$ by using constraint (3.11c). The remaining one cycle is obtained by employing natural variable y_e which is a binary variable whose value is one if edge $e \in E$ is in the cycle, and zero otherwise. The constraints (3.11b) are extended two matching constraints which ensure that each node $i \in V$ is covered by either exactly one cycle from $\mathscr{J}_t$ or exactly two edges that are incident to node i. Constraints (3.11d) ensure that at least one extra undirected cycle (not in $\mathscr{J}_t$) is selected. Note that the variables y_e may form more than one cycle yielding a solution to Problem 3.11 having more than p cycles. This situation will be discussed later in this subsection

and again in 3.3.3.2.

As mentioned earlier, we adopt the third search strategy which entails combining the individual R_t^{p-1} into one resolution problem, R^{p-1}. The feasible region of each R_t^{p-1} is a subset of the feasible region of R^{p-1}. The collective resolution problem R^{p-1} can then be formulated as follows:

$$
\begin{aligned}
(R^{p-1}) \quad \text{minimize} \quad & \sum_{\forall e \in E} d_e y_e + \sum_{k \in \mathscr{J}} c_k x_k && && \text{(3.12a)} \\
\text{subject to} \quad & \sum_{\forall e \in \delta(i)} y_e + \sum_{k \in \mathscr{J}} 2a_{ik} x_k = 2, && \forall i \in V && \text{(3.12b)} \\
& \sum_{k \in \mathscr{J}} x_k = p - 1 && && \text{(3.12c)} \\
& \sum_{e \in E} y_e \geq 3 && && \text{(3.12d)} \\
& x_k \in \{0, 1\} && \forall k \in \mathscr{J} && \text{(3.12e)} \\
& y_e \in \{0, 1\} && \forall e \in E && \text{(3.12f)}
\end{aligned}
$$

Clearly, the only difference between Problems 3.11 and 3.12 is in extending the cycles x_k to include all the cycles that are non-zero in the optimal LP solution of all the exploration nodes solved in the main branch. The feasible region of R^{p-1} contains all the optimal solutions of each individual R_t^{p-1}, if one exists. Since the variables y_e may form more than one cycle in the optimal solution of R^{p-1}, Algorithm 5 is used to guarantee that we get exactly p cycles (i.e., one cycle via y_e's and $p-1$ cycles from $\mathscr{J}$).

Algorithm 5

1: **Input:** $G = (V, E)$, a cost $d_{ij}, \forall (i,j) \in E$, set of cycles $\mathscr{J}$ with lengths $c_k, \forall k \in \mathscr{J}$, and a positive number p.
2: **Output:** Upper bound for HpMP for the given value of p.
3: **repeat**
4: Solve Problem 3.12. Let y_e^* and x_k^* be its optimal solution.
5: Let $\mathscr{Y} := \{e \in E | y_e^* = 1\}$ and $\mathscr{X} := \{k \in \mathscr{J} | x_k^* = 1\}$
6: Let NC be the number of cycles formed by the edges in $\mathscr{Y}$.
7: Add $\sum_{e \in \mathscr{Y}} y_e \leq |\mathscr{Y}| - 1$ to Problem 3.12.
8: **until** $NC = 1$
9: **return** The $p-1$ cycles in $\mathscr{X}$ and the single cycle formed by the edges in $\mathscr{Y}$.

3.3.3.2 Resolution Procedure for $L = p - 3$

In this case, the resolution node t will thus contain the constraint $\sum_{k \in \mathscr{J}_t} x_k \geq L+1 = p-2$. We refer to these resolution problems as R_t^{p-2}. Again, as we adopt the third search strategy, we combine the individual R_t^{p-2} to form one merged resolution problem, R^{p-2}. We divide solving the resolution problem into three steps. In the first step, we solve the resolution problem which includes the constraint $\sum_{k \in \mathscr{J}_t} x_k = p$; whereas in the second step, we solve the resolution problem which includes the constraint $\sum_{k \in \mathscr{J}_t} x_k = p-1$; and finally, in the third step, we explain how the resolution problem (which includes the constraint $\sum_{k \in \mathscr{J}_t} x_k = p-2$) is formulated and solved. The first two steps were explained in Section 3.3.3.1. Next, we provide the details of the mathematical formulation and the algorithm for the third step.

The collective resolution problem R^{p-2} can be formulated as follows:

$$(R^{p-2}) \quad \text{minimize} \sum_{\forall e \in E} \sum_{m=1}^{m=2} d_{em} y_{em} + \sum_{k \in \mathscr{J}} c_k x_k \tag{3.13a}$$

$$\text{subject to} \sum_{\forall e \in \delta(i)} y_{e1} + \sum_{k \in \mathscr{J}} 2a_{ik}x_k = 2z_i, \qquad \forall i \in V \tag{3.13b}$$

$$\sum_{\forall e \in \delta(i)} y_{e2} = 2(1 - z_i), \qquad \forall i \in V \tag{3.13c}$$

$$\sum_{k \in \mathscr{J}} x_k = p - 2 \tag{3.13d}$$

$$\sum_{e \in E} y_{em} \geq 3, \qquad \forall m \in \{1, 2\} \tag{3.13e}$$

$$x_j \in \{0, 1\} \qquad \forall j \in \mathscr{J} \tag{3.13f}$$

$$y_{em} \in \{0, 1\} \qquad \forall e \in E, m = 1, 2 \tag{3.13g}$$

We devise Algorithm 6 (given below) to guarantee that exactly p cycles are eventually obtained when solving R^{p-2}.

Algorithm 6 Solving R^{p-2}

Input: $G = (V, E)$, a cost $d_{e1} = d_{e2} = d_e, \forall e \in E$, set of cycles $\mathscr{J}$ with lengths $c_k, \forall k \in \mathscr{J}$, and a positive number p.
Output: Upper bound for HpMP for the given value of p.
repeat
  Solve Problem 3.13. Let y^*_{e1}, y^*_{e2}, and x^*_k be its optimal solution.
  Let $\mathscr{Y}_1 := \{e \in E | y^*_{e1} = 1\}$, $\mathscr{Y}_2 := \{e \in E | y^*_{e2} = 1\}$ and $\mathscr{X} := \{k \in \mathscr{J} | x^*_k = 1\}$
  Let NC_1 and NC_2 be the number of cycles formed by the edges in $\mathscr{Y}_1$ and $\mathscr{Y}_2$, resp.
  if $NC_1 \neq 1$ **then**
    Add $\sum_{e \in \mathscr{Y}_1} y_{e1} \leq |\mathscr{Y}_1| - 1$ and $\sum_{e \in \mathscr{Y}_1} y_{e2} \leq |\mathscr{Y}_1| - 1$ to Problem 3.13.
  end if
  if $NC_2 \neq 1$ **then**
    Add $\sum_{e \in \mathscr{Y}_2} y_{e1} \leq |\mathscr{Y}_2| - 1$ and $\sum_{e \in \mathscr{Y}_2} y_{e2} \leq |\mathscr{Y}_2| - 1$ to Problem 3.13.
  end if
until ($NC_1 = 1$ and $NC_2 = 1$)
return The $p - 2$ cycles in $\mathscr{X}$, the single cycle via $\mathscr{Y}_1$, and the single cycle via $\mathscr{Y}_2$.

3.4 Computational Results

This section presents the computational results of the proposed SBP method when used in solving the HpMP. We specifically compared the performance of the SBP method when $L = p - 2$ and $L = p - 3$ to that of the B&P algorithm implemented in Chapter 2. The algorithm was tested on 18 complete graphs from the TSPLIB available from `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/`. The size of the selected graphs ranges from 58 nodes to 200 nodes. The edge costs are rounded to the nearest integer as is the common convention in the TSP literature.

All algorithms were run on a machine with an Intel Core i7 processor and 32 GB of memory. Implementations were coded and compiled on Visual Studio C++ using standard template library and standard subroutines. Linear and integer programs were solved using CPLEX 12.4 invoked in C++ using Concert Technology. The time limit for all the test instances is set to one hour.

Tables 3.1-3.3 show the running times (in seconds) for the three aforementioned algorithms when solving HpMP. In case an algorithm failed to find an optimal solution within the one hour time limit, the optimality gap (in percentage) is then reported. The optimality gap (OG) is defined as $OG = \frac{BFS-LB}{BFS} * 100\%$ where BFS is the best feasible solution and LB is the lower bound. As discussed earlier, the feasible solution is provided via solving exactly the resolution nodes. In the case where $L = p - 3$, the merged resolution problem is, in general, more difficult to solve than when $L = p - 2$ and the time limit may be reached without finishing solving the resolution problem. In this case, by relaxing constraint (3.13d) to be a greater than or equal inequality and solving Problem 3.13, we obtain a lower bound for the resolution problem. The minimum of the lower bound provided by the last

exploration node and the lower bound provided by the relaxation of the resolution problem yields a valid lower bound for HpMP.

In all the tables in this chapter, the following convention is adopted:

1. The columns with heading **graph-p** represent the graph name followed by the required number of cycles, p.

2. The columns with heading **B&P** present the computational results for the B&P algorithm presented in Chapter 2.

3. The columns with heading **SBP$^{\mathbf{p-2}}$** present the computational results for the SBP method when $L = p - 2$.

4. The columns with heading **SBP$^{\mathbf{p-3}}$** present the computational results for the SBP method when $L = p - 3$.

Tables 3.1-3.3 show the computational results for the 333 test instances. For any instance, the blue, purple, and red numbers represent the best, the second best, and the worst performing algorithm, respectively. We only provide computational results for the instances with $p > p_{2m} + 3$ where p_{2m} is the number of cycles in the minimum weight two matching problem. This is because, as shown in Section 2.3.3, using the column generation framework is not recommended when solving HpMP for $p \leq p_m + 3$ due to the increasing difficulty of the pricing problem. Therefore, since the core of SBP is the column generation framework and our main objective is to compare the computational performance of our proposed slim branching tree and that of the traditional branching tree in B&P, we focused on the p values that are greater than $p_{2m} + 3$ in our computational experiments.

We next give some detailed comparison between each pair of the three compared algorithms. In Sections 3.4.1 and 3.4.2, we compare the performance of B&P and

the SBP method when $L = p - 2$ and $L = p - 3$, respectively. In Section 3.4.3, we compare the performance of SBP when $L = p-2$ and SBP when $L = p-3$. Finally, Section 3.4.4 presents the performance profile for the three tested algorithms.

3.4.1 Comparison of B&P and SBP$^{p\text{-}2}$

Tables 3.1-3.3 show that SBP$^{\text{p-2}}$ performed better in 80.2% of the instances; whereas B&P performed better in 9.3% of the instances. Both algorithms had the same computational times (or OG) in 10.81% of the instances (of these instances, 92% provided an integer solution when solving the LP relaxation at the root node—since the two algorithms use the same procedure to solve the root node, the two algorithms had the same computational times).

SBP$^{\text{p-2}}$ found the optimal solution for 66 instances (i.e., 19.8%) that B&P failed to solve to optimality within the one hour time limit; whereas B&P succeeded in accomplishing that in only two instances. For the instances that both algorithms failed to solve to optimality within the time limit, the average OG for SBP$^{\text{p-2}}$ was 0.53%; whereas B&P has an average OG of 2.42%.

SBP$^{\text{p-2}}$ was at least two times faster than B&P in 32.4% of the instances; at least three times faster than B&P in 21.6% of the instances, at least four times faster than B&P in 15.3% of the instances; and at least five times faster in 12.3% of the instances.

3.4.2 Comparison of B&P and SBP$^{p\text{-}3}$

Tables 3.1-3.3 show that SBP$^{\text{p-3}}$ performed better than B&P in 76.6% of the instances; whereas B&P performed better in 13.5% of the instances. Both algorithms had the same computational times (or OG) in 9.9% of the instances; except for one instance, these are the instances that are solved at the root node without any branching required.

SBP$^{p\text{-}3}$ found the optimal solution for 90 instances (i.e., 27%) that B&P failed to solve to optimality within the one hour time limit; whereas B&P found the optimal solution for 6 instances (i.e., 1.8%) that SBP$^{p\text{-}3}$ failed to solve to optimality. For the instances that both algorithms failed to solve to optimality within the time limit, the average OG for SBP$^{p\text{-}3}$ was 0.98%; whereas B&P has an average OG of 2.93%.

SBP$^{p\text{-}3}$ was at least two times faster than B&P in 39.9% of the instances; at least three times faster than B&P in 30% of the instances, at least four times faster than B&P in 21% of the instances; and at least five times faster in 14.1% of the instances.

3.4.3 Comparison of SBP$^{p\text{-}2}$ and SBP$^{p\text{-}3}$

Tables 3.1-3.3 show that SBP$^{p\text{-}3}$ performed better than SBP$^{p\text{-}2}$ in 51.4% of the instances; whereas SBP$^{p\text{-}2}$ performed better in 39% of the instances. Both algorithms had the same computational times (or OG) in 9.6% of the instances.

SBP$^{p\text{-}3}$ found the optimal solution for 25 instances (i.e., 7.51%) that SBP$^{p\text{-}2}$ failed to solve to optimality within the one hour time limit; whereas SBP$^{p\text{-}2}$ solved only seven instances that are unsolved by SBP$^{p\text{-}3}$ in the one hour time limit. For the instances that both algorithms failed to solve to optimality within the time limit, the average OG for SBP$^{p\text{-}3}$ was 0.99%; whereas SBP$^{p\text{-}2}$ has an average OG of 0.66%.

SBP$^{p\text{-}3}$ was at least two times faster than SBP$^{p\text{-}2}$ in 5.7% of the instances; and at least three times faster than SBP$^{p\text{-}2}$ in 2.1% of the instances.

3.4.4 Performance Profile

Figure 3.6 presents the performance profile for B&P, SBP$^{p\text{-}2}$, and SBP$^{p\text{-}3}$. The performance profile is constructed based on the approach proposed in [17]. Specifically, let n_i be the number of instances, $\mathscr{P}$ be the set of instances, n_a be the number of algorithms studied, and $\mathscr{A}$ be the set of algorithms. For each $i \in \mathscr{P}$ and $a \in \mathscr{A}$,

Table 3.1: Computational Results for graphs brazil58, eil76, pr76, gr96, rat99, rd100, kroa100, and krob100. (Solution times in seconds or optimality gaps in percentage: the best in blue, the second best in purple, and the worst in red.)

graph-p	B&P	SBP$^{p\text{-}2}$	SBP$^{p\text{-}3}$
brazil58-15	50	38	37
brazil58-16	91	42	38
brazil58-17	305	57	40
brazil58-18	223	61	56
brazil58-19	198	94	77
st70-16	12	12	12
st70-17	37	32	48
st70-18	36	31	36
st70-19	116	28	41
st70-20	94	33	36
st70-21	456	54	55
st70-22	2348	129	158
st70-23	2.1%	717	383
eil76-6	425	155	105
eil76-7	141	93	96
eil76-8	0.2%	218	215
eil76-9	35	35	35
eil76-10	77	77	77
eil76-11	63	63	63
eil76-12	92	92	92
eil76-13	166	96	116
eil76-14	66	56	71
eil76-15	124	72	88
eil76-16	256	104	72
eil76-17	70	70	70
eil76-18	197	99	94
eil76-19	79	104	78
eil76-20	1005	92	105
eil76-21	37	37	37
eil76-22	34	34	34
eil76-23	31	31	31
eil76-24	1021	87	113
eil76-25	1140	110	175
pr76-11	2719	2115	627
pr76-12	0.1%	3072	849
pr76-13	0.1%	3584	881
pr76-14	0.3%	0.18%	961
pr76-15	0.6%	0.33%	1005
pr76-16	146	146	146
pr76-17	190	252	211
pr76-18	108	244	209
pr76-19	184	175	221
pr76-20	29	29	29
pr76-21	38	38	38
pr76-22	30	30	30
pr76-23	169	150	92
pr76-24	829	195	144
pr76-25	359	137	115

graph-p	B&P	SBP$^{p\text{-}2}$	SBP$^{p\text{-}3}$
gr96-13	0.8%	697	1143
gr96-14	350	895	1603
gr96-15	0.1%	1708	1491
gr96-16	0.2%	0.11%	1156
gr96-17	1.7%	3531	827
gr96-18	3%	392	617
gr96-19	501	314	405
gr96-20	405	128	157
gr96-21	483	144	155
gr96-22	3488	323	338
gr96-23	0.2%	737	392
gr96-24	0.2%	1008	633
gr96-25	0.1%	675	578
gr96-26	1667	543	373
gr96-27	1809	244	406
gr96-28	0.4%	1804	1135
gr96-29	1.6%	0.32%	2670
gr96-30	2.6%	0.56%	2886
gr96-31	5%	0.67%	0.66%
gr96-32	4.5%	1.5%	1.6%
rat99-9	168	609	294
rat99-10	146	107	284
rat99-11	350	379	594
rat99-12	744	381	316
rat99-13	467	373	241
rat99-14	423	398	258
rat99-15	925	540	300
rat99-16	459	395	287
rat99-17	1932	716	376
rat99-18	681	422	278
rat99-19	484	264	206
rat99-20	833	255	218
rat99-21	135	127	232
rat99-22	956	146	207
rat99-23	509	187	265
rat99-24	445	168	299
rat99-25	533	166	222
rat99-26	415	166	232
rat99-27	740	209	247
rat99-28	1009	269	297
rat99-29	1045	167	230
rat99-30	1199	187	283
rat99-31	377	214	220
rat99-32	3062	181	237
rat99-33	0.5%	597	438

graph-p	B&P	SBP$^{p\text{-}2}$	SBP$^{p\text{-}3}$
rd100-17	148	117	207
rd100-18	260	502	244
rd100-19	3529	1504	850
rd100-20	0.4%	840	599
rd100-21	705	447	437
rd100-22	0.1%	290	245
rd100-23	252	188	205
rd100-24	164	141	91
rd100-25	51	51	51
rd100-26	349	206	190
rd100-27	456	208	221
rd100-28	590	302	183
rd100-29	486	382	257
rd100-30	306	263	240
rd100-31	0.1%	612	319
rd100-32	0.1%	298	282
rd100-33	3.1%	0.86%	0.89%
kroa100-17	107	105	102
kroa100-18	209	257	275
kroa100-19	331	601	540
kroa100-20	427	298	211
kroa100-21	112	182	149
kroa100-22	809	406	212
kroa100-23	1140	821	431
kroa100-24	545	260	275
kroa100-25	129	248	149
kroa100-26	325	457	267
kroa100-27	1444	786	661
kroa100-28	1428	1089	554
kroa100-29	1355	1060	524
kroa100-30	2500	1158	662
kroa100-31	0.5%	1943	976
kroa100-32	2.4%	0.67%	0.68%
kroa100-33	2.7%	1.9%	2%
krob100-23	0.1%	136	184
krob100-24	57	57	57
krob100-25	109	141	125
krob100-26	0.2%	219	231
krob100-27	138	116	127
krob100-28	48	48	48
krob100-29	48	48	48
krob100-30	124	63	120
krob100-31	883	170	197
krob100-32	3483	297	332
krob100-33	2.8%	1.8%	2%

Table 3.2: Computational Results for graphs kroc100, kroe100, lin105, gr120, bier127, and u159. (Solution times in seconds or optimality gaps in percentage: the best in blue, the second best in purple, and the worst in red.)

graph-p	B&P	SBP^{p-2}	SBP^{p-3}
kroc100-16	3600	0.31%	1905
kroc100-17	0.12%	1206	563
kroc100-18	436	209	189
kroc100-19	2281	313	291
kroc100-20	0.2%	519	390
kroc100-21	0.47%	635	480
kroc100-22	0.14%	452	315
kroc100-23	1574	236	242
kroc100-24	0.01%	340	303
kroc100-25	0.15%	917	625
kroc100-26	0.3%	1790	1125
kroc100-27	0.29%	2462	1168
kroc100-28	0.25%	1539	1149
kroc100-29	0.3%	2262	933
kroc100-30	2.26%	1283	816
kroc100-31	2.8%	1474	659
kroc100-32	1.91%	2262	785
kroc100-33	6.07%	2.25%	2.7%
kroe100-15	454	198	221
kroe100-16	538	394	261
kroe100-17	0.04%	1242	415
kroe100-18	3547	398	215
kroe100-19	250	156	153
kroe100-20	114	73	144
kroe100-21	97	97	97
kroe100-22	331	133	185
kroe100-23	2017	417	380
kroe100-24	531	299	267
kroe100-25	3479	354	220
kroe100-26	547	159	142
kroe100-27	1930	354	232
kroe100-28	1178	275	250
kroe100-29	468	127	162
kroe100-30	380	231	199
kroe100-31	1640	177	272
kroe100-32	0.24%	414	366
kroe100-33	2.6%	47	52
lin105-23	588	259	200
lin105-24	733	274	292
lin105-25	0.3%	1099	476
lin105-26	147	126	144
lin105-27	43	43	43
lin105-28	1863	300	334
lin105-29	1022	149	248
lin105-30	0.4%	473	321
lin105-31	788	229	251
lin105-32	0.4%	830	971
lin105-33	1.5%	0.14%	2826
lin105-34	3.4%	0.61%	2858
lin105-35	4.7%	0.61%	0.55%

graph-p	B&P	SBP^{p-2}	SBP^{p-3}
gr120-19	542	3547	2732
gr120-20	1062	1478	1051
gr120-21	110	110	110
gr120-22	97	97	97
gr120-23	256	248	287
gr120-24	1919	470	483
gr120-25	134	134	134
gr120-26	125	125	125
gr120-27	764	389	465
gr120-28	2229	1609	817
gr120-29	1004	1362	855
gr120-30	481	1067	677
gr120-31	602	594	551
gr120-32	883	510	646
gr120-33	562	366	554
gr120-34	87	87	87
gr120-35	108	108	108
gr120-36	1677	360	443
gr120-37	1424	362	427
gr120-38	0.1%	509	405
gr120-39	0.7%	1109	2071
gr120-40	2.1%	1262	1073
bier127-15	0.1%	912	786
bier127-16	86	86	86
bier127-17	262	350	312
bier127-18	177	167	198
bier127-19	291	280	247
bier127-20	590	543	333
bier127-21	156	156	313
bier127-22	476	475	362
bier127-23	95	95	95
bier127-24	104	104	104
bier127-25	232	375	376
bier127-26	101	101	101
bier127-27	125	125	125
bier127-28	101	101	101
bier127-29	218	218	218
bier127-30	1374	630	502
bier127-31	826	498	389
bier127-32	432	515	406
bier127-33	0.1%	1123	1023
bier127-34	2.3%	0.43%	0.44%
bier127-35	13%	0.72%	0.66%
bier127-36	11%	0.38%	0.37%
bier127-37	2.1%	0.4%	0.42%
bier127-38	2.2%	0.37%	0.38%
bier127-39	5.3%	0.06%	1364
bier127-40	0.2%	1921	0.25%

graph-p	B&P	SBP^{p-2}	SBP^{p-3}
u159-24	0.06%	1135	706
u159-25	0.08%	842	706
u159-26	3381	1122	677
u159-27	0.01%	1803	871
u159-28	0.18%	1270	732
u159-29	0.15%	1288	738
u159-30	0.01%	1211	984
u159-31	0.04%	2154	1363
u159-32	0.1%	0.02%	2311
u159-33	0.19%	2547	1946
u159-34	0.03%	1477	1208
u159-35	0.09%	1818	1311
u159-36	0.31%	2405	1318
u159-37	0.13%	2480	1873
u159-38	0.04%	1388	863
u159-39	829	903	733
u159-40	0.04%	1024	792
u159-41	0.27%	1631	910
u159-42	0.44%	1906	2094
u159-43	0.26%	2596	1830
u159-44	0.63%	0.17%	0.22%
u159-45	2.65%	0.25%	0.44%
u159-46	0.43%	0.38%	0.39%
u159-47	1%	0.65%	0.71%
u159-48	0.88%	0.56%	0.67%
u159-49	2.28%	0.93%	1.12%
u159-50	3.56%	1%	1.12%
u159-51	3.4%	1.85%	1.93%
u159-52	12.9%	1.97%	2.34%
u159-53	8.47%	3.65%	4.25%

Table 3.3: Computational Results for graphs kroa150, kroa200, and krob200. (Solution times in seconds or optimality gaps in percentage: the best in blue, the second best in purple, and the worst in red.)

graph-p	B&P	SBP$^{p\text{-}2}$	SBP$^{p\text{-}3}$	graph-p	B&P	SBP$^{p\text{-}2}$	SBP$^{p\text{-}3}$
kroa150-23	0.36%	0.21%	3540	krob200-41	0.04%	0.16%	0.25%
kroa150-24	0.24%	0.08%	2230	krob200-42	0.08%	0.20%	0.31%
kroa150-25	2.86%	0.25%	0.23%	krob200-43	0.01%	0.11%	0.19%
kroa150-26	0.50%	0.21%	0.09%	krob200-44	0.14%	0.06%	0.57%
kroa150-27	0.36%	0.07%	3200	krob200-45	0.02%	0.06%	0.66%
kroa150-28	0.33%	0.23%	0.2%	krob200-46	0.08%	0.06%	0.19%
kroa150-29	0.25%	0.21%	0.07%	krob200-47	0.04%	0.04%	0.19%
kroa150-30	0.39%	0.07%	2494	krob200-48	0.09%	0.10%	0.19%
kroa150-31	0.18%	0.01%	2486	krob200-49	0.04%	0.04%	2853
kroa150-32	0.09%	3067	2159	krob200-50	3522	3354	3150
kroa150-33	0.06%	2773	3130	krob200-51	0.04%	0.02%	0.17%
kroa150-34	0.11%	0.03%	2639	krob200-52	1503	502	1610
kroa150-35	0.05%	2390	1586	krob200-53	2640	2298	0.18%
kroa150-36	0.11%	2938	2009	krob200-54	2513	2377	0.85%
kroa150-37	0.14%	2587	2746	krob200-55	3027	2944	0.85%
kroa150-38	1.44%	0.06%	2232	krob200-56	2549	1548	0.27%
kroa150-39	0.18%	0.15%	0.12%	krob200-57	3489	1354	0.62%
kroa150-40	1.29%	0.25%	0.45%	krob200-58	3378	0.14%	0.38%
kroa150-41	1.73%	0.20%	0.31%				
kroa150-42	1.17%	0.13%	0.19%				
kroa150-43	0.17%	0.07%	0.21%				
kroa150-44	1.06%	0.06%	0.95%				
kroa150-45	1.26%	0.19%	0.51%				
kroa150-46	2.36%	0.42%	0.63%				
kroa150-47	4.89%	0.65%	0.77%				
kroa150-48	2.18%	1.05%	1.22%				
kroa150-49	7.35%	2.93%	2.85%				
kroa150-50	8.71%	3.82%	3.89%				
kroa200-40	0.95%	0.13%	3236				
kroa200-41	0.12%	1.04%	3422				
kroa200-42	0.16%	0.04%	0.09%				
kroa200-43	0.07%	0.13%	2358				
kroa200-44	0.03%	0.09%	1843				
kroa200-45	0.08%	3170	0.2%				
kroa200-46	0.11%	0.10%	2590				
kroa200-47	0.48%	0.21%	0.19%				
kroa200-48	1.21%	0.03%	3220				
kroa200-49	0.08%	0.03%	2887				
kroa200-50	0.28%	0.09%	0.03%				
kroa200-51	1.79%	0.07%	0.56%				
kroa200-52	0.67%	0.02%	0.18%				
kroa200-53	0.48%	0.56%	0.21%				
kroa200-54	2.06%	0.08%	0.2%				
kroa200-55	0.93%	0.17%	0.55%				
kroa200-56	1.36%	1.67%	0.55%				
kroa200-57	2.46%	0.25%	2.89%				
kroa200-58	8.57%	0.41%	8.74%				
kroa200-59	6.19%	0.72%	2.22%				
kroa200-60	23.93%	0.64%	2.85%				

define the performance ratio as

$$r_{i,a} = \frac{t_{i,a}}{\min\{t_{i,a} : a \in \mathscr{A}\}} \tag{3.14}$$

where $t_{i,a}$ is a performance measure. In our case, the performance measure is the computational time if an instance is solved to optimality by at least one of the algorithms within the one hour time limit; the computational time is set to 3600 if an algorithm failed to solve an instance. If all the studied algorithms failed to solve an instance to optimality, the performance measure is the OG. Finally, define

$$\rho_a(\tau) = \frac{|\{i \in \mathscr{P}\} : r_{i,a} \leq \tau\}|}{n_i} * 100\% \tag{3.15}$$

as the percentage of instances that an algorithm $a \in \mathscr{A}$ has a performance ratio that is within $\tau \in \mathbb{R}$ of the best possible ratio.

The top graph in Figure 3.6 presents the performance profile for the three algorithms for the instances that are solved to optimality by at least one algorithm. The top graph in Figure 3.6 shows that SBP$^{\text{p-3}}$ has the best performance in 65.8% of the test instances. SBP$^{\text{p-2}}$ has the second best performance and was the best algorithm in 40.4%. Finally, B&P comes last and it was the best algorithm in 17.6% of the instances. Tables 3.1-3.3 show that this 17.6% (in which B&P has good performance) is occurring in the instances that are solved at the root node. For these instances, all the three algorithms have the exact performance.

When the value of τ is greater that four, the performances of SBP$^{\text{p-2}}$ and SBP$^{\text{p-3}}$ are almost identical. In other words, the computational time required to solve any instance using SBP$^{\text{p-2}}$ is at most four times the computational time required to solve the same instance using SBP$^{\text{p-3}}$. Clearly, SBP$^{\text{p-2}}$ and SBP$^{\text{p-3}}$ performed better than

B&P.

The bottom graph in Figure 3.6 presents the performance profile for the three algorithms for the instances that are not solved to optimality by any of the three algorithms. The bottom graph in Figure 3.6 shows that SBP$^{p\text{-}2}$ has the best performance in 68.9% of the test instances. SBP$^{p\text{-}3}$ has the second best overall performance and was the best algorithm in 23%. Finally, B&P comes last and it was the best algorithm in only 9.8% of the instances.

The reason for the better OG provided by SBP$^{p\text{-}2}$ over those provided by SBP$^{p\text{-}3}$ for the unsolved instances is next explained. In one hand, in SBP$^{p\text{-}3}$, after exhausting the one hour in solving the exploration nodes, only the relaxation of the combined resolution problem R^{p-2} was solved in order to provide a valid lower bound. On the other hand, in SBP$^{p\text{-}2}$, even after exhausting the one hour in solving the exploration nodes, R^{p-1} was solved to optimality in all these instances. This is due to the fact that R^{p-1} (used in SBP$^{p\text{-}2}$) can be solved in at most five seconds; whereas solving R^{p-2} (used in SBP$^{p\text{-}3}$) can take considerably longer time.

In summary, we conclude that SBP$^{p\text{-}3}$ has the best overall performance. It is important to stress that there is no practical difference in OG provided by SBP$^{p\text{-}3}$ and SBP$^{p\text{-}2}$ (i.e., 0.33%) for the instances that are not solved to optimality by both algorithms.

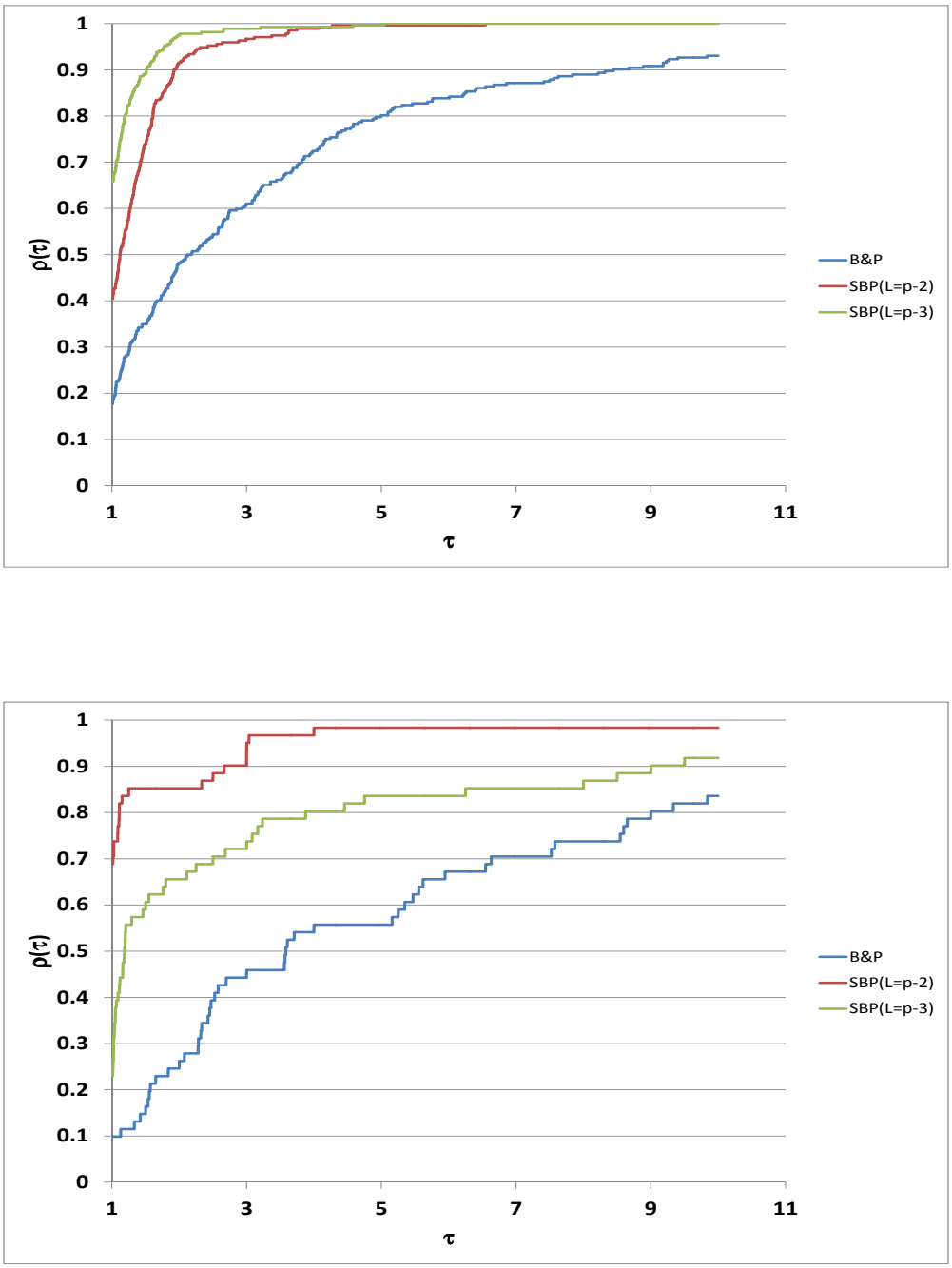


Figure 3.6: Performance profiles for B&P, SBP^{p-2}, SBP^{p-3}: the top figure shows the performance profile using the computational times using the instances which at least one algorithm solved to optimality; the bottom figure shows the performance profile using the optimality gaps using the instances which all algorithms failed to solve to optimality.

4. REDUCED COST FIXING IN SBP WITH APPLICATION TO HPMP

In recent years, one of the reasons that helped in improving the performance of integer program solvers was the efficient implementation of preprocessing techniques. These techniques include deleting redundant constraints, changing variable bounds, changing the coefficients in the constraint matrix, among other techniques (e.g. [43], [27], [47]). In this chapter, we focus on the reduced cost fixing technique which is used to fix the values of a subset of variables optimally in any binary program in a preprocessing stage [37]. In order to implement reduced cost fixing, we need to obtain a feasible solution to the problem at hand and a tight lower bound. In this chapter, we explain how reduced cost fixing can be used to enhance the performance of SBP when solving the HpMP.

As the first ingredient in reduced cost fixing, NVM (natural variable space model) provides the tightest LP relaxation *lower bound* for HpMP among the seven studied formulations [21]. However, NVM formulation has exponential number of constraints which are difficult to separate. While the separation of these constraints is NP-hard as shown in [21], we circumvent this difficulty by only separating a subset of these constraints that performs efficiently in practice.

The second ingredient needed to implement reduced cost fixing is a good feasible solution as an *upper bound*. We develop a new effective heuristic based on k-opt moves to find a good feasible solution for HpMP.

The following lemma from [37] formalizes the reduced cost fixing technique.

Lemma 4.0.1 (Reduced cost fixing)**.** *Let $\bar{c}_{ij}$ be the reduced cost of nonbasic x_{ij} obtained after solving the LP relaxation of NVM, also let Z_{LP} and Z^* be the lower and upper bounds of NVM, respectively. If $Z_{LP} + \bar{c}_{ij} > Z^*$, then set $x_{ij} = 0$.*

Based on our experimental results when solving the HpMP, we noticed that HpMPs on complete graphs are more difficult to solve and require more computational time than similar size problems on incomplete graphs. Thus, deleting the edges that cannot be in an optimal solution has a good impact on the time required to solve HpMP. For this purpose, we propose the use of reduced cost fixing to delete such edges [37]. In our setting, this technique can be used to delete some variables by fixing the values of some decision variables to zero.

After getting a tighter lower bound by implementing the separation algorithms presented in Section 4.1 and a good feasible solution by implementing k-opt as explained in Section 4.2, one can implement reduced cost fixing presented in Lemma 4.0.1. This implies setting the values of the edges having reduced cost value that is greater than the difference between the upper and lower bounds to zero, thus deleting such edges from the graph. Then, SBP method presented in Section 3.3 can be implemented on the resulting graph. In Section 4.3, we present the computational results for the SBP with reduced cost fixing technique.

4.1 Lower Bound for HpMP

Before we present the generalized natural variable space model (GNVM) which is used to find the lower bound used in reduced cost fixing, we first define the notation used. Let P be a partition of set of vertices V into m subsets given as $\{S_1, \ldots, S_m\}$ and let $\mathscr{P}_m^l$ be the set of partitions of size m such that the cardinality of each $S_i, i = 1, \ldots, m$ is greater than or equal to l. For a partition P, define E_P as the set of edges straddling pairs of subsets S_v and S_w (i.e., $E_P = \{(i,j) \in E : i \in S_v, j \in S_w; S_v \neq S_w; S_v, S_w \in P\}$), and $\mathscr{C}_p = \{C \subset E : |C| = |V|$ and the edges in C form at most p cycles$\}$.

Now, the mathematical formulation of GNVM is given by:

$$\textbf{(GNVM)} \quad \text{minimize } H = \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{4.1a}$$

$$\text{subject to} \sum_{\forall(i,j)\in\delta(v)} x_{ij} = 2 \qquad \forall v \in V \tag{4.1b}$$

$$\sum_{(i,j)\in E_P} x_{ij} \geq 1 + u \qquad \forall P \in \mathscr{P}^3_{p+u} \tag{4.1c}$$

$$\sum_{(i,j)\notin C} x_{ij} \geq 1 + u \qquad \forall C \in \mathscr{C}_{p-u} \tag{4.1d}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in E \tag{4.1e}$$

In this model, x_{ij} equals one if edge (i,j) is in the optimal solution, and is zero otherwise. In the constraints (4.1b), $\delta(v)$ denotes the set of edges incident to node $v \in V$, these constraints imply that exactly two of the edges incident to v must be selected.

Constraints (4.1c) enforce that, for each partition P of graph G with $p+u$ subsets, at least $1+u$ edges from E_P should be selected in order to eventually obtain the target p cycles. These constraints prevent the formation of more than p cycles. For example, if the number of cycles in the solution is four and the number of required cycles is two, then the sum of the edges in E_P is greater than or equal to three. Observe that these constraints are the generalization of the cut set constraints for the TSP which can be obtained by setting $p = 1$ in (4.1c).

Constraints (4.1d) ensure that, for each partition of graph G into $p-u$ subsets, at least $1+u$ additional edges are needed in any feasible solution to get p cycles. These constraints prevent the formation of less than p cycles. For example, if the number of cycles in the solution is two and the number of required cycles is five, then the sum of the edges not in C (i.e., all the edges E after excluding the edges forming

these two cycles) is greater than or equal to four.

In [21], the authors proved that NVM has the tightest LP bound at the root node, they also showed that identifying the most violated constraints (4.1c) and (4.1d) is NP-hard. Thus, finding a separation algorithm for these constraints is challenging. Although, the authors in [21] provide separation algorithms for constraints (4.1c) and (4.1d), they concluded that the performance of their separation algorithms was not satisfactory. Therefore, we develop two new efficient separation algorithms for constraints (4.1c) and (4.1d) which are critical to obtain a tighter lower bound, Z_{LP}.

The idea in the two new separation algorithms is to identify the partitions of V having more or less than p cycles that violate constraints (4.1c) and (4.1d), respectively. The first separation algorithm, presented in Algorithm 7, requires solving TSP using LKH heuristic in [23]; whereas the second separation algorithm, presented in Algorithm 8, entails solving the minimum weight 2-matching problem [18].

Algorithm 7 LKH separation

1: **Input:** $G = (V, E)$, a cost $d_{ij}, \forall e \in E$, and a positive number p.
2: **Output:** Lower bound for HpMP for the given value of p, and the reduced cost for each $(i, j) \in E$.
3: **repeat**
4: Solve the LP relaxation of GNVM to get the optimal solution x^*_{ij}.
5: Construct $G^* = (V, E)$ with edge costs $d_{ij} = -x^*_{ij}$.
6: Run LKH heuristic on G^* to get the tour C.
7: Add the constraint $\sum_{(i,j)\notin C} x_{ij} \geq p$ to GNVM.
8: **until** $\sum_{(i,j)\notin C} x^*_{ij} \geq p$
9: **return** Optimal LP solution of GNVM and the reduced cost of each edge $(i, j) \in E$.

Separation Algorithm 7 is based on LKH heuristic which can be used to find a

feasible TSP tour (i.e., the output of LKH is one cycle). Thus, the value of $p-u$ is one in constraints (4.1d) and can be written as $\sum_{(i,j)\notin C} x_{ij} \geq p, \forall C \in \mathscr{C}_1$. The idea in this algorithm is to find the minimum value of the left hand side of the latter inequality. This is equivalent to finding the maximum value of $\sum_{(i,j)\in C} x_{ij}$ since $\sum_{(i,j)\in C} x_{ij} = |E| - \sum_{(i,j)\notin C} x_{ij}$. This is achieved by using LKH heuristic to find the *minimum* weight TSP tour on G^* with edge costs $d_{ij} = -x^*_{ij}$.

Algorithm 8 Two-matching separation

1: **Input:** $G = (V, E)$, a cost $d_{ij}, \forall e \in E$, and a positive number p.
2: **Output:** Lower bound for HpMP for the given value of p, and the reduced cost for each $(i, j) \in E$.
3: **while** True **do**
4: Solve the LP relaxation of GNVM to get the optimal solution x^*_{ij}.
5: Construct $G^* = (V, E)$ with edge costs $d_{ij} = -x^*_{ij}$.
6: Solve the minimum weight two-matching problem on G^* to get p_{2m} cycles.
7: Use the minimum weight two-matching optimal solution to identify E_P or C.
8: **if** $p_{2m} > p$ and $\sum_{(i,j)\in E_P} x^*_{ij} < 1 + p_{2m} - p$ **then**
9: Add the constraint $\sum_{(i,j)\in E_P} x_{ij} \geq 1 + p_{2m} - p$ to GNVM.
10: **else if** $p_{2m} < p$ and $\sum_{(i,j)\notin C} x^*_{ij} < 1 + p - p_{2m}$ **then**
11: Add the constraint $\sum_{(i,j)\notin C} x_{ij} \geq 1 + p - p_{2m}$ to GNVM.
12: **else break**
13: **end if**
14: **end while**
15: **return** Optimal LP solution of GNVM and the reduced cost of each edge $(i, j) \in E$.

Separation Algorithm 8 is based on the minimum weight two-matching solution which is used to find p_{2m} cycles whose sum of costs is minimum. Based on the value of p_{2m}, we may have either a violated constraint (4.1c) or a violated constraint (4.1d).

On one hand, if p_{2m} is greater than p, then we may have a violated constraint (4.1c). In order to test whether there exists a violated constraint, we start by replacing $p+u$ by p_{2m} in constraints (4.1c) to get $\sum_{(i,j)\in E_P} x_{ij} \geq 1 + p_{2m} - p, \forall P \in \mathscr{P}^3_{p_{2m}}$. Next, we want to find the minimum value of the left hand side of the latter inequality. This is equivalent to finding the maximum value of $\sum_{(i,j)\notin E_P} x_{ij}$ since $\sum_{(i,j)\in E_P} x_{ij} = |E| - \sum_{(i,j)\notin E_P} x_{ij}$. This is achieved by using the two-matching algorithm to find the minimum weight two-matching on G^* with edge costs $d_{ij} = -x^*_{ij}$.

On the other hand, if p_{2m} is less than p, then we may have a violated constraint (4.1d). In order to test whether there exists a violated constraint, we start by replacing $p-u$ by p_{2m} in constraints (4.1d) to get $\sum_{(i,j)\notin C} x_{ij} \geq 1+p-p_{2m}, \forall C \in \mathscr{C}_{p_{2m}}$. Next, we want to find the minimum value of the left hand side of the latter inequality. This is equivalent to finding the maximum value of $\sum_{(i,j)\in C} x_{ij}$ since $\sum_{(i,j)\in C} x_{ij} = |E| - \sum_{(i,j)\notin C} x_{ij}$. This is achieved by using the two-matching algorithm to find the minimum weight two-matching on G^* with edge costs $d_{ij} = -x^*_{ij}$.

4.2 Upper Bound for HpMP

Our heuristic is based on k-opt local search algorithm as motivated by the successful implementation of k-opt for solving the TSP reported in [23] and [31]. We first give a definition of k-opt moves for HpMP and then present how k-opt can be used to find good feasible solutions for HpMP for different values of p.

Definition 4.2.1 (k-opt for HpMP)**.** *Let F be a set of q cycles, $q \neq p$, that partition the graph $G = (V, E)$. Then, k-opt$^+$ (k-opt$^-$) for HpMP exchanges k edges from F by another k edges in $E \setminus F$ in such a way that q is increased (or decreased) by exactly one cycle.*

Note that this definition is slightly different than that presented in TSP literature since HpMP has an extra parameter, p, which has to be considered. For k-opt heuris-

tic, an algorithm that quickly provides a starting solution, F, is needed. Moreover, in our computational study presented in Chapter 2 to solve HpMP, we observed that the optimal solutions for p and $p+1$ cycles differ in only a small subset of edges. That is, the solutions that deviate from the required number of cycles may still contain majority of the edges in the optimal solution. Thus, k-opt heuristic seems promising to convert an initial feasible solution to one that is close to optimal.

We employ two efficient algorithms to obtain two starting solutions. The first one is blossomV which provides the optimal two-matching solution in polynomial time [28] . The second one is LKH algorithm which provides a TSP solution [23]. Both algorithms are very fast and require less than one second to solve complete graphs with 200 nodes.

Based on the starting solution, we now can implement k-opt$^+$ (k-opt$^-$) moves to increase (or decrease) the number of cycles by one iteratively until reaching the target number of cycles, p. Specifically, on one hand, if the starting feasible solution has p_{2m} cycles (i.e., the 2-matching optimal solution), the k-opt$^+$ or k-opt$^-$ is applied $|p - p_{2m}|$ times to get a feasible solution with p cycles. On the other hand, if the starting feasible solution has one cycle (i.e., TSP), then the k-opt$^+$ is applied $p - 1$ times to increase the number of cycles to p.

Clearly, different algorithms are needed for different values of k in k-opt. A straightforward implementation of k-opt, based on exhaustive enumeration, has complexity $O(|V|^k)$. This complexity can be improved by discarding some cases that cannot increase (or decrease) the number of cycles by exactly one as discussed later. This improvement results in a significant speed up of the algorithm compared to exhaustive enumeration. Next, we give an overview of the 2-opt$^+$ and 2-opt$^-$ algorithms by emphasizing the different cases arising from deleting exactly two edges, and then discarding the impossible subsets of cases.

2-opt$^+$: It is applied when p is greater than the starting number of cycles. There are two cases when deleting two edges from a solution, namely, either the two edges lie on the same cycle or on two different cycles. Here, the second case is not possible since no two edges can then be added to increase the number of cycles. Thus, the second case is discarded and this improves the computational time considerably.

2-opt$^-$: It is applied when p is less than the starting number of cycles. Again, there are two cases, either the two edges lie on the same cycle or on two different cycles. Here, the first case is not possible since no two edges can then be added to decrease the number of cycles by one. Thus, the first case is discarded.

Clearly, more cases are expected when implementing the 3-opt moves. Since we focus only on instances where $p > p_{2m} + 3$, we next explain the three main cases observed in 3-opt$^+$. Specifically, these include: (i) one edge is deleted from three different cycles, (ii) one edge is deleted from a cycle and two edges are deleted from another cycle, and (iii) three edges are deleted from the same cycle.

As shown in Figure 4.1, in the first case, we cannot form four cycles by deleting an edge from three different cycles. Thus, this case is discarded. Also, in the second case, as shown in Figure 4.2, we cannot get three cycles by deleting two edges from one cycle and an edge from another cycle by using three new (i.e., edges that do not form any of the changed cycles) edges. Thus, the second case can also be discarded. Note that the only way to form three cycles in this case is by reusing the single deleted edge from a cycle (i.e., the deleted edge in the rightmost cycle in Figure 4.2). Based on the previous discussion, the only case that should be considered is the third case. Hence, discarding the first two cases allowed us to implement 3-opt$^+$ more efficiently.

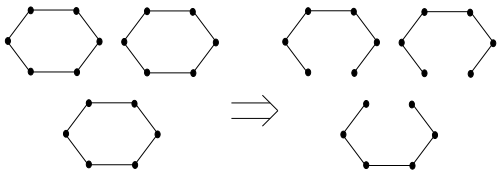


Figure 4.1: Case (i): one edge is deleted from three different cycles.

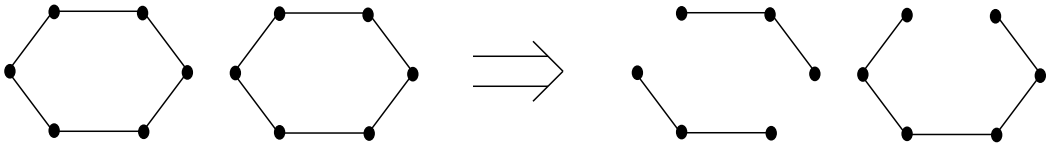


Figure 4.2: Case (ii): two edges are deleted from one cycle and one edge from another.

Similar ideas can be used to speed up the higher k-opt moves although the number of possible cases considerably increases. In practice, the computational times for 2-opt, 3-opt, and 4-opt were very satisfactory.

From our preliminary tests, the closeness of p to p_{2m} or one is the most important factor in selecting which starting feasible solution to use. In other words, if p is closer to p_{2m} than to one, then the 2-matching solution, in general, provides better feasible solution for HpMP with p cycles, and vice versa. Our preliminary tests show that the two aforementioned starting solutions (i.e., TSP heuristic and two-matching optimal solution) give satisfactory results.

4.3 Computational Results

This section presents the computational results of the proposed SBP method after applying the reduced cost fixing technique when solving the HpMP. We refer

to this algorithm as $\text{SBP}_{+}^{\text{p-2}}$. We specifically compare the performance of the $\text{SBP}_{+}^{\text{p-2}}$ method to the B&P algorithm presented in Chapter 2 and to $\text{SBP}^{\text{p-2}}$ algorithm presented in Chapter 3. The algorithm was tested on 18 complete graphs from the TSPLIB available from `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/`. The size of the selected graphs ranges from 58 nodes to 200 nodes. The edge costs are rounded to the nearest integer as is the common convention in the TSP literature.

All algorithms were run on a machine with an Intel Core i7 processor and 32 GB of memory. Linear and integer programs were solved using CPLEX 12.4 invoked in C++ using Concert Technology. The time limit for all the test instances is set to one hour.

Tables 4.1-4.3 show the computational running times in seconds when solving HpMP for the three aforementioned algorithms. In case an algorithm failed to find an optimal solution within the one hour time limit, the optimality gap (in percentage) is then reported. The optimality gap (OG) is defined as $OG = \frac{BFS-LB}{BFS} * 100\%$ where BFS is the best feasible solution and LB is the lower bound. For any instance, the blue, purple, and red numbers represent the best, the second best, and the worst performing algorithm, respectively.

In all the tables in this chapter, the following convention is adopted:

1. The columns with heading **graph-p** represent the graph name followed by the required number of cycles, p.

2. The columns with heading **B&P** present the computational results for the B&P algorithm presented in Chapter 2.

3. The columns with heading $\mathbf{SBP^{p-2}}$ present the computational results for the SBP method when $L = p - 2$ presented in Chapter 3.

4. The columns with heading $\mathbf{SBP_{+}^{p\text{-}2}}$ present the computational results for the SBP method when $L = p - 2$ and the reduced cost fixing technique is implemented.

Sections 4.3.1 presents a comparison of the performance of B&P and the $\text{SBP}_{+}^{\text{p-2}}$ method; whereas Section 4.3.2 presents a comparison of the performance of $\text{SBP}^{\text{p-2}}$ and $\text{SBP}_{+}^{\text{p-2}}$. Finally, Section 4.3.3 presents the performance profile for the three tested algorithms.

4.3.1 Comparison of B&P and $SBP_{+}^{p\text{-}2}$

Tables 4.1-4.3 show that $\text{SBP}_{+}^{\text{p-2}}$ performed better than B&P in 94.6% of the instances; whereas B&P performed better in 5.1% of the instances. Both algorithms had the same computational times in one instance.

$\text{SBP}_{+}^{\text{p-2}}$ found the optimal solution for 87 instances (i.e., 26.1% of the instances) that B&P failed to solve to optimality within the one hour time limit; whereas B&P never found the optimal solution if $\text{SBP}_{+}^{\text{p-2}}$ failed to find it within the one hour time limit. For the instances that both algorithms failed to solve to optimality within the time limit, the average OG for $\text{SBP}_{+}^{\text{p-2}}$ was 0.55%; whereas B&P has an average OG of 2.89%.

$\text{SBP}_{+}^{\text{p-2}}$ was at least two times faster than B&P in 52.9% of the instances; at least three times faster than B&P in 40.8% of the instances, at least four times faster than B&P in 32.7% of the instances; and at least five times faster in 25.8% of the instances.

4.3.2 Comparison of $SBP^{p\text{-}2}$ and $SBP_{+}^{p\text{-}2}$

Tables 4.1-4.3 show that $\text{SBP}_{+}^{\text{p-2}}$ performed better than $\text{SBP}^{\text{p-2}}$ in 88.3% of the instances; whereas $\text{SBP}^{\text{p-2}}$ performed better in 9.31% of the instances. Both algorithms had the same computational times in eight instances.

$SBP_{+}^{p\text{-}2}$ found the optimal solution for 21 instances (i.e., 6.3% of the instances) that $SBP^{p\text{-}2}$ failed to solve to optimality within the one hour time limit; whereas $SBP^{p\text{-}2}$ succeeded in accomplishing that in only one instance. For the instances that both algorithms failed to solve to optimality within the time limit, the average OG for $SBP_{+}^{p\text{-}2}$ was 0.54%; whereas $SBP^{p\text{-}2}$ has an average OG of 0.64%.

$SBP_{+}^{p\text{-}2}$ was at least two times faster than $SBP^{p\text{-}2}$ in 30% of the instances; at least three times faster than $SBP^{p\text{-}2}$ in 12.9% of the instances, at least four times faster than $SBP^{p\text{-}2}$ in 6.6% of the instances; and at least five times faster in 3% of the instances.

4.3.3 Performance Profile

Figures 4.3 presents the performance profile for the three tested algorithms. The same technique explained in Chapter 3 is used to construct the performance profiles. Specifically, the top graph in Figure 4.3 shows the performance profile for B&P, $SBP^{p\text{-}2}$, and $SBP_{+}^{p\text{-}2}$ using the instances that are solved to proven optimality by at least one of the aforementioned algorithms. In this case, the performance measure $t_{i,a}$ is taken to be the computational time (in seconds). If an algorithm failed to find the instance optimal solution within one hour, we set $t_{i,a}$ to 3600. In contrast, the bottom graph in Figure 4.3 shows the performance profile for B&P, $SBP^{p\text{-}2}$, and $SBP_{+}^{p\text{-}2}$ using the instances that the three algorithms failed to solve to proven optimality within one hour. In this case, the performance measure $t_{i,a}$ is taken to be OG.

The top graph in Figure 4.3 shows that $SBP_{+}^{p\text{-}2}$ has the best performance and it was the best algorithm in 91% of the instances. $SBP^{p\text{-}2}$ has the second best performance and it was the best algorithm in 5% of the instances. Finally, B&P comes last and it was the best algorithm in only 4% of the instances.

Table 4.1: Computational Results for graphs brazil58, eil76, pr76, gr96, rat99, rd100, kroa100, and krob100. (Solution times in seconds or optimality gaps in percentage: the best in blue, the second best in purple, and the worst in red.)

graph-p	B&P	SBP$^{p\text{-}2}$	SBP$^{p\text{-}2}_{+}$	graph-p	B&P	SBP$^{p\text{-}2}$	SBP$^{p\text{-}2}_{+}$	graph-p	B&P	SBP$^{p\text{-}2}$	SBP$^{p\text{-}2}_{+}$
brazil58-15	50	38	12	gr96-13	0.8%	697	613	rd100-17	148	117	38
brazil58-16	91	42	21	gr96-14	350	895	250	rd100-18	260	502	280
brazil58-17	305	57	29	gr96-15	0.1%	1708	1346	rd100-19	3529	1504	875
brazil58-18	223	61	35	gr96-16	0.2%	0.11%	3512	rd100-20	0.4%	840	734
brazil58-19	198	94	77	gr96-17	1.7%	3531	2467	rd100-21	705	447	383
st70-16	12	12	11	gr96-18	3%	392	304	rd100-22	0.1%	290	246
st70-17	37	32	11	gr96-19	501	314	200	rd100-23	252	188	95
st70-18	36	31	10	gr96-20	405	128	81	rd100-24	164	141	63
st70-19	116	28	12	gr96-21	483	144	88	rd100-25	51	51	47
st70-20	94	33	18	gr96-22	3488	323	218	rd100-26	349	206	180
st70-21	456	54	29	gr96-23	0.2%	737	646	rd100-27	456	208	182
st70-22	2348	129	90	gr96-24	0.2%	1008	894	rd100-28	590	302	284
st70-23	2.1%	717	657	gr96-25	0.1%	675	628	rd100-29	486	382	453
eil76-6	425	155	21	gr96-26	1667	543	284	rd100-30	306	263	134
eil76-7	141	93	32	gr96-27	1809	244	203	rd100-31	0.1%	612	292
eil76-8	0.2%	218	46	gr96-28	0.4%	1804	1149	rd100-32	0.1%	298	251
eil76-9	35	35	22	gr96-29	1.6%	0.32%	0.05%	rd100-33	3.1%	0.86%	0.6%
eil76-10	77	77	24	gr96-30	2.6%	0.56%	0.62%	kroa100-17	107	105	43
eil76-11	63	63	16	gr96-31	5%	0.67%	0.65%	kroa100-18	209	257	118
eil76-12	92	92	19	gr96-32	4.5%	1.5%	1.61%	kroa100-19	331	601	253
eil76-13	166	96	34	rat99-9	168	609	278	kroa100-20	427	298	93
eil76-14	66	56	53	rat99-10	146	107	298	kroa100-21	112	182	103
eil76-15	124	72	62	rat99-11	350	379	143	kroa100-22	809	406	194
eil76-16	256	104	52	rat99-12	744	381	51	kroa100-23	1140	821	361
eil76-17	70	70	41	rat99-13	467	373	75	kroa100-24	545	260	149
eil76-18	197	99	59	rat99-14	423	398	73	kroa100-25	129	248	119
eil76-19	79	104	61	rat99-15	925	540	205	kroa100-26	325	457	183
eil76-20	1005	92	63	rat99-16	459	395	159	kroa100-27	1444	786	314
eil76-21	37	37	32	rat99-17	1932	716	337	kroa100-28	1428	1089	477
eil76-22	34	34	22	rat99-18	681	422	177	kroa100-29	1355	1060	506
eil76-23	31	31	29	rat99-19	484	264	140	kroa100-30	2500	1158	319
eil76-24	1021	87	86	rat99-20	833	255	126	kroa100-31	0.5%	1943	2241
eil76-25	1140	110	149	rat99-21	135	127	80	kroa100-32	2.4%	0.67%	0.81%
pr76-11	2719	2115	1304	rat99-22	956	146	73	kroa100-33	2.7%	1.9%	1.66%
pr76-12	0.1%	3072	2264	rat99-23	509	187	89	krob100-23	0.1%	136	221
pr76-13	0.1%	3584	3514	rat99-24	445	168	106	krob100-24	57	57	25
pr76-14	0.3%	0.18%	0.18%	rat99-25	533	166	106	krob100-25	109	141	36
pr76-15	0.6%	0.33%	0.33%	rat99-26	415	166	122	krob100-26	0.2%	219	73
pr76-16	146	146	122	rat99-27	740	209	89	krob100-27	138	116	109
pr76-17	190	252	213	rat99-28	1009	269	193	krob100-28	48	48	42
pr76-18	108	244	191	rat99-29	1045	167	187	krob100-29	48	48	42
pr76-19	184	175	163	rat99-30	1199	187	213	krob100-30	124	63	51
pr76-20	29	29	22	rat99-31	377	214	128	krob100-31	883	170	153
pr76-21	38	38	30	rat99-32	3062	181	213	krob100-32	3483	297	215
pr76-22	30	30	30	rat99-33	0.5%	597	429	krob100-33	2.8%	1.8%	0.81%
pr76-23	169	150	84								
pr76-24	829	195	131								
pr76-25	359	137	154								

Table 4.2: Computational Results for graphs kroc100, kroe100, lin105, gr120, bier127, and u159. (Solution times in seconds or optimality gaps in percentage: the best in blue, the second best in purple, and the worst in red.)

graph-p	B&P	SBP^{p-2}	SBP$^{p-2}_{+}$
kroc100-16	3600	0.31%	3548
kroc100-17	0.12%	1206	945
kroc100-18	436	209	130
kroc100-19	2281	313	187
kroc100-20	0.2%	519	305
kroc100-21	0.47%	635	429
kroc100-22	0.14%	452	439
kroc100-23	1574	236	208
kroc100-24	0.01%	340	287
kroc100-25	0.15%	917	490
kroc100-26	0.3%	1790	782
kroc100-27	0.29%	2462	859
kroc100-28	0.25%	1539	1018
kroc100-29	0.3%	2262	772
kroc100-30	2.26%	1283	1021
kroc100-31	2.8%	1474	1237
kroc100-32	1.91%	2262	2377
kroc100-33	6.07%	2.25%	2.23%
kroe100-15	454	198	41
kroe100-16	538	394	226
kroe100-17	0.04%	1242	731
kroe100-18	3547	398	288
kroe100-19	250	156	116
kroe100-20	114	73	24
kroe100-21	97	97	27
kroe100-22	331	133	91
kroe100-23	2017	417	232
kroe100-24	531	299	172
kroe100-25	3479	354	212
kroe100-26	547	159	102
kroe100-27	1930	354	198
kroe100-28	1178	275	168
kroe100-29	468	127	71
kroe100-30	380	231	57
kroe100-31	1640	177	77
kroe100-32	0.24%	414	217
kroe100-33	2.6%	47	97
lin105-23	588	259	57
lin105-24	733	274	61
lin105-25	0.3%	1099	658
lin105-26	147	126	24
lin105-27	43	66	18
lin105-28	1863	300	58
lin105-29	1022	149	35
lin105-30	0.4%	473	103
lin105-31	788	229	59
lin105-32	0.4%	830	240
lin105-33	1.5%	0.14%	0.08%
lin105-34	3.4%	0.61%	0.26%
lin105-35	4.7%	0.61%	0.62%

graph-p	B&P	SBP^{p-2}	SBP$^{p-2}_{+}$
gr120-19	542	3547	3529
gr120-20	1062	1478	1052
gr120-21	110	275	81
gr120-22	97	247	93
gr120-23	256	282	138
gr120-24	1919	470	286
gr120-25	134	225	101
gr120-26	125	239	112
gr120-27	764	389	235
gr120-28	2229	1609	106
gr120-29	1004	1362	670
gr120-30	481	1067	502
gr120-31	602	594	296
gr120-32	883	510	413
gr120-33	562	366	321
gr120-34	87	364	232
gr120-35	108	224	228
gr120-36	1677	360	351
gr120-37	1424	362	313
gr120-38	0.1%	509	500
gr120-39	0.7%	1109	1220
gr120-40	2.1%	1262	1843
bier127-15	0.1%	912	95
bier127-16	86	252	46
bier127-17	262	350	116
bier127-18	177	754	100
bier127-19	291	529	115
bier127-20	590	543	144
bier127-21	156	371	131
bier127-22	476	475	146
bier127-23	95	221	65
bier127-24	104	194	74
bier127-25	232	375	170
bier127-26	101	195	95
bier127-27	125	199	86
bier127-28	101	213	106
bier127-29	218	204	101
bier127-30	1374	630	459
bier127-31	826	498	354
bier127-32	432	515	187
bier127-33	0.1%	1123	1105
bier127-34	2.3%	0.43%	0.48%
bier127-35	13%	0.72%	0.63%
bier127-36	11%	0.38%	0.38%
bier127-37	2.1%	0.4%	0.42%
bier127-38	2.2%	0.37%	0.31%
bier127-39	5.3%	0.06%	0.06%
bier127-40	0.2%	1921	1881

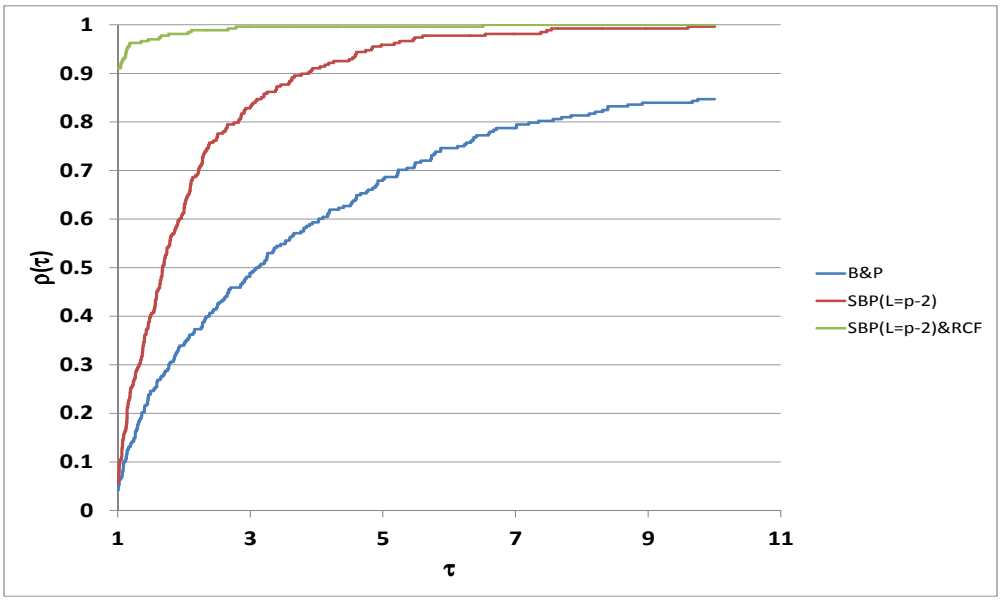
graph-p	B&P	SBP^{p-2}	SBP$^{p-2}_{+}$
u159-24	0.06%	1135	248
u159-25	0.08%	842	204
u159-26	3381	1122	332
u159-27	0.01%	1803	322
u159-28	0.18%	1270	444
u159-29	0.15%	1288	513
u159-30	0.01%	1211	468
u159-31	0.04%	2154	1357
u159-32	0.1%	0.02%	2097
u159-33	0.19%	2547	1895
u159-34	0.03%	1477	795
u159-35	0.09%	1818	957
u159-36	0.31%	2405	1553
u159-37	0.13%	2480	1118
u159-38	0.04%	1388	691
u159-39	829	903	655
u159-40	0.04%	1024	566
u159-41	0.27%	1631	688
u159-42	0.44%	1906	1332
u159-43	0.26%	2596	1232
u159-44	0.63%	0.17%	2768
u159-45	2.65%	0.25%	0.04%
u159-46	0.43%	0.38%	0.16%
u159-47	1%	0.65%	0.23%
u159-48	0.88%	0.56%	0.28%
u159-49	2.28%	0.93%	0.82%
u159-50	3.56%	1%	0.95%
u159-51	3.4%	1.85%	1.52%
u159-52	12.9%	1.97%	2.07%
u159-53	8.47%	3.65%	3.45%

Table 4.3: Computational Results for graphs kroa150, kroa200, and krob200. (Solution times in seconds or optimality gaps in percentage: the best in blue, the second best in purple, and the worst in red.)

graph-p	B&P	SBP$^{p\text{-}2}$	SBP$^{p\text{-}2}_{+}$	graph-p	B&P	SBP$^{p\text{-}2}$	SBP$^{p\text{-}2}_{+}$
kroa150-23	0.36%	0.21%	0.05%	krob200-41	0.04%	0.16%	0.10%
kroa150-24	0.24%	0.08%	0.01%	krob200-42	0.08%	0.20%	0.19%
kroa150-25	2.86%	0.25%	0.15%	krob200-43	0.01%	0.11%	0.21%
kroa150-26	0.50%	0.21%	0.11%	krob200-44	0.14%	0.06%	0.03%
kroa150-27	0.36%	0.07%	0.07%	krob200-45	0.02%	0.06%	3574
kroa150-28	0.33%	0.23%	0.16%	krob200-46	0.08%	0.06%	0.02%
kroa150-29	0.25%	0.21%	0.08%	krob200-47	0.04%	0.04%	0.03%
kroa150-30	0.39%	0.07%	0.01%	krob200-48	0.09%	0.10%	0.10%
kroa150-31	0.18%	0.01%	0.11%	krob200-49	0.04%	0.04%	<0.01%
kroa150-32	0.09%	3067	2965	krob200-50	3522	3354	2444
kroa150-33	0.06%	2773	2678	krob200-51	0.04%	0.02%	0.02%
kroa150-34	0.11%	0.03%	0.02%	krob200-52	1503	502	395
kroa150-35	0.05%	2390	2250	krob200-53	2640	2298	2048
kroa150-36	0.11%	2938	1749	krob200-54	2513	2377	2157
kroa150-37	0.14%	2587	1876	krob200-55	3027	2944	2765
kroa150-38	1.44%	0.06%	2005	krob200-56	2549	1548	1354
kroa150-39	0.18%	0.15%	2895	krob200-57	3489	1354	1140
kroa150-40	1.29%	0.25%	0.03%	krob200-58	3378	0.14%	0.11%
kroa150-41	1.73%	0.20%	3519				
kroa150-42	1.17%	0.13%	2406				
kroa150-43	0.17%	0.07%	2303				
kroa150-44	1.06%	0.06%	2192				
kroa150-45	1.26%	0.19%	3361				
kroa150-46	2.36%	0.42%	0.29%				
kroa150-47	4.89%	0.65%	0.66%				
kroa150-48	2.18%	1.05%	1.10%				
kroa150-49	7.35%	2.93%	3.41%				
kroa150-50	8.71%	3.82%	3.81%				
kroa200-40	0.95%	0.13%	1581				
kroa200-41	0.12%	1.04%	2553				
kroa200-42	0.16%	0.04%	0.21%				
kroa200-43	0.07%	0.13%	1259				
kroa200-44	0.03%	0.09%	2453				
kroa200-45	0.08%	3170	0.16%				
kroa200-46	0.11%	0.17%	0.25%				
kroa200-47	0.48%	0.21%	2563				
kroa200-48	1.21%	0.03%	2175				
kroa200-49	0.08%	0.03%	2471				
kroa200-50	0.28%	0.09%	<0.01%				
kroa200-51	1.79%	0.07%	3185				
kroa200-52	0.67%	0.02%	<0.01%				
kroa200-53	0.48%	0.56%	0.01%				
kroa200-54	2.06%	0.08%	3340				
kroa200-55	0.93%	0.17%	0.12%				
kroa200-56	1.36%	1.67%	0.25%				
kroa200-57	2.46%	0.25%	0.26%				
kroa200-58	8.57%	0.41%	0.33%				
kroa200-59	6.19%	0.72%	0.44%				
kroa200-60	23.93%	0.64%	0.85%				

Again, for the instances that the three algorithms failed to solve to optimality, the bottom graph in Figure 4.3 shows that $SBP_{+}^{p\text{-}2}$ has the best performance and it was the best algorithm in 71% of the instances. $SBP^{p\text{-}2}$ has the second best performance and it was the best algorithm in 31% of the instances. Finally, B&P comes last and it was the best algorithm in only 8% of the instances.

In summary, we conclude that $SBP_{+}^{p\text{-}2}$ has the best overall performance among the three tested algorithms for all the test instances.



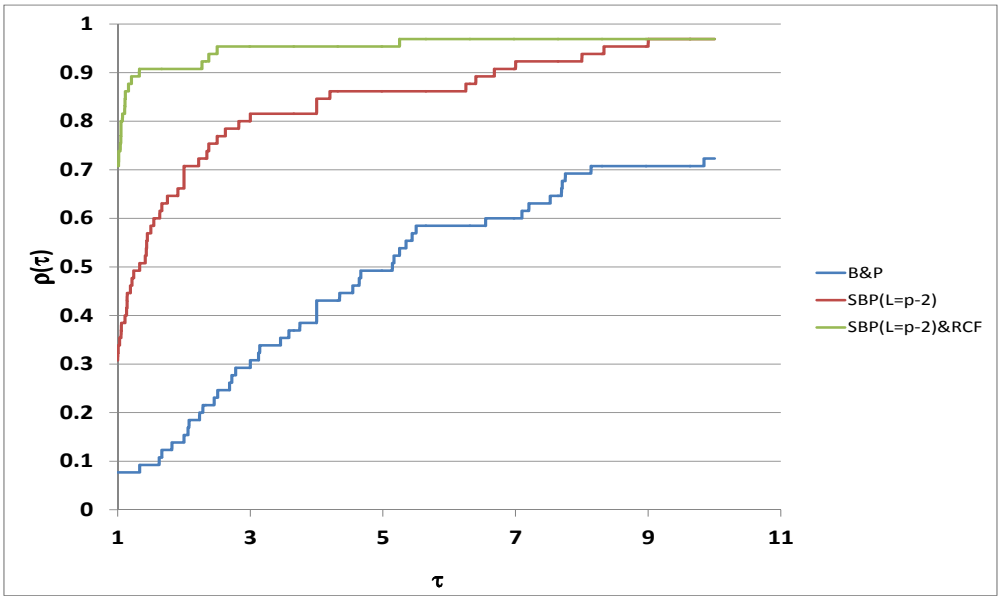


Figure 4.3: Performance profiles for B&P, SBP$^{p\text{-}2}$, SBP$^{p\text{-}2}_{+}$: the top figure shows the performance profile using the computational times using the instances which at least one algorithm solved to optimality; the bottom figure shows the performance profile using the optimality gaps using the instances which all algorithms failed to solve to optimality.

5. CONCLUDING REMARKS

5.1 Summary of Contributions and Conclusions

The main objective of this dissertation is to present a new exact optimization method, the Slim Branch and Price (SBP) method, which is an improvement over the traditional B&P framework. SBP can be used to solve a large class of combinatorial optimization problems that can be solved by B&P type algorithms and that have binary master problems with fixed support (i.e., the sum of the variables in any feasible solution is fixed). This is an important class of problems as it includes several classical and fundamental problems. We tested our proposed method on an interesting problem that falls into the aforementioned category. This problem is known as HpMP and is a generalization of the well-known TSP.

In order to test SBP, since there was no B&P algorithm presented in the literature to solve HpMP, we started by developing a B&P algorithm to solve HpMP, then we presented the SBP method and tested it on the same instances of HpMP, finally, we used reduced cost fixing preprocessing technique to improve the performance of SBP.

In Chapter 2 of the dissertation, we developed a B&P algorithm to solve the HpMP and compared our computational results to those of the NVM presented in [21]. Several contributions on modeling, methodology, and computational aspects were also presented. Specifically, 1) we modified the set partitioning formulation of HpMP proposed by [21]; 2) we developed a new efficient algorithm to find the shortest cycle in an undirected graph with arbitrary edge costs and no negative cycles; 3) we developed an algorithm to find the most negative cycle in an undirected graph with arbitrary edge costs; 4) computationally, the proposed algorithm for solving the HpMP outperformed the previously presented algorithms as it successfully solves in-

stances up to 318 nodes, as opposed to other exact algorithms which solved instances up to 100 nodes; 5) we proved that for every value of p, the HpMP is NP-hard even when restricted to Euclidean graphs; and 6) we showed that the practical complexity of HpMP and the performance of the algorithms to solve it substantially depend on the relation between p and p_{2m} (the number of cycles in the 2-matching optimal solution), furthermore, we were able to explain the reason for the good performance of B&P when p is greater than $p_{2m} + 3$ and the reason for the good performance of NVM when p is between $p_{2m} - 3$ and $p_{2m} + 3$, inclusive.

The comparison of the computational results of our B&P algorithm and NVM from [21] presents an interesting strategy when solving the HpMP. We start by solving the minimum weight two-matching problem to find p_{2m}. If the value of the required cycles p is close to p_{2m}, (i.e., $p_{2m} - 3 \leq p \leq p_{2m} + 3$), using the NVM presented in [21] is recommended. If $p > p_{2m} + 3$, it is much faster to solve HpMP using the proposed B&P algorithm. Both algorithms perform poorly, especially in larger instances, whenever $p < p_{2m} - 3$ and further research is needed to solve these instances.

Finally, we note that [30] presented a variant of HpMP in which p is the upper limit on the number of required cycles. Here we define a new variant of HpMP in which the number of cycles is required to be at least p. Our B&P algorithm can be used to solve both of these HpMP variants. Interestingly, when solving the newly defined variant, our algorithms are guaranteed to solve the pricing problem in polynomial time. This is because by changing the equality constraint (2.1c) to a greater than or equal constraint, the dual variable of the modified constraint is always nonnegative, and therefore only cases 2 and 3 arise when solving the pricing problem.

In Chapter 3 of the dissertation, we presented the Slim Branch and Price (SBP)

method. The main advantage in SBP is that the branching tree has only one main branch with several leaves. We refer to the nodes forming the main branch as exploration nodes and to the leaf nodes as resolution nodes. The main function of the exploration nodes is to explore new portions of the feasible regions effectively; whereas the main function of the resolution nodes is to find efficiently optimal integer solutions in the close neighborhood of the current LP solution.

SBP can be interpreted as a branching framework or as a cutting plane framework. As a branching framework, the core idea of the SBP method is to improve the traditional branching scheme of B&P with the objective of exploring the problem's feasible region more efficiently and effectively. As a cutting plane framework, after obtaining the optimal solution of the LP relaxation at any node, SBP adds a linear inequality (i.e., exploration inequality) that is violated by this optimal LP solution. However, this cut is invalid because it may excise a subset of the feasible region that may contain the optimal integer solution. Adding a sequence of these aggressive (but invalid) cuts results in excising several feasible regions that may contain the optimal solution(s). To guarantee the exactness of SBP, we generate a resolution problem whose feasible region is the union of the aforementioned excised feasible regions.

The parameter L, used to construct the exploration inequalities, plays an important role in the performance of SBP and it balances the aggressiveness of the exploration inequality and the easiness of the resolution problem. The value of L is primarily selected to guarantee the easiness of the resolution problem. In general, we recommend selecting L values that is close (but not extremely close) to the value of the fixed support, p. We presented two implementations of SBP using $L = p - 2$ and $L = p - 3$ when solving HpMP. We refer to these implementations as SBP$^{\text{p-2}}$ and SBP$^{\text{p-3}}$, respectively.

We showed that SBP$^{\text{p-3}}$ has the best performance and it was the best algorithm in

62% of the instances among the three tested algorithms (B&P, SBP$^{p\text{-}2}$, and SBP$^{p\text{-}3}$). SBP$^{p\text{-}2}$ has the second best performance and it was the best algorithm in 45% of the instances. Finally, B&P comes last and it was the best algorithm in only 18% of the instances. Note that the majority of this 18% (in which B&P had good performance) was occurring in the instances that were solved at the root node. For these instances, all the three algorithms had the exact performance and this explains why the percentages do not add up to 100%.

In Chapter 4, we implemented reduced cost fixing to improve the performance of SBP. Reduced cost fixing is a preprocessing technique that helps in reducing the size of the problem by fixing optimally the values of a subset of the problem variables. In order to fix these values, we need to calculate lower and upper bounds to HpMP.

In order to calculate a lower bound to HpMP, we developed two efficient separation algorithms for a subset of constraints in the generalized natural variable space model for HpMP. These separation algorithms provided us with better lower bounds than those reported in [21].

In order to calculate an upper bound for HpMP, we developed a new heuristic based on k-opt moves to obtain good feasible solutions for HpMP. This was motivated by the good performance of k-opt in TSP literature [23]. The quality of the feasible solutions was very good and the implementation of k-opt was very fast. After calculating the lower and upper bounds, reduced cost fixing was implemented and helped in deleting a considerable number of edges.

We illustrated the computational advantage of SBP with reduced cost fixing, which we refer to as SBP$_+$, over SBP without reduced cost fixing on the HpMP. Specifically, we compared the performance of SBP$_+^{p\text{-}2}$ and SBP$^{p\text{-}2}$ by fixing the value of L to $p-2$. As expected, SBP$_+^{p\text{-}2}$ provided better computational times in most of the test instances. It also allowed us to solve to optimality several instances that

$SBP^{p\text{-}2}$ failed to solve in the one hour time limit.

We also showed that $SBP_{+}^{p\text{-}2}$ has the best performance and it was the best algorithm in 87% of the instances among the three tested algorithms. $SBP^{p\text{-}2}$ has the second best performance and it was the best algorithm in only 11% of the instances. Finally, B&P comes last and it was the best algorithm in only 5% of the instances.

It is important to note that SBP method performed better than B&P in spite of changing the structure of the pricing problem used in B&P. Specifically, recall that we showed in Chapter 2 that the pricing problem in B&P when solving HpMP for $p > p_{2m} + 3$ was solved using polynomial algorithms except for rare occasions when case 4 is invoked. But, after adding the exploration inequalities, these efficient algorithms are no longer valid. Instead, the pricing problem was always solved as an IP model in SBP. Despite this fact, the performance of SBP was much better than that of B&P as the advantage of adding the exploration inequalities surpassed the increased complexity of the pricing problem.

In summary, we showed that B&P can solve to optimality HpMP instances with up to 127 nodes; whereas the NVM presented in [21] solved instances with up to 40 nodes within the one hour time limit. Next, we illustrated the computational advantage of SBP over B&P when solving HpMP. In particular, within one hour time limit, SBP can solve to optimality instances with up to 200 nodes; whereas B&P can solve to optimality instances with up to 127 nodes. Finally, we showed that the reduced cost fixing technique can greatly enhance the performance of SBP.

5.2 Future Research

We plan to use SBP to solve other problems such as parallel machine scheduling [4] and its variants, capacitated p-median problem [32] and its variants, balanced disjoint rings problem [45], k-clustering problem [22], and political districting problem [34].

The main challenge in this line of research is modifying the pricing problems used in B&P to prevent the formation of certain columns. Another challenge is finding a formulation for the resolution problem in order to achieve good computational results.

Another interesting direction is generalizing SBP to solve any master problem by relaxing the condition of fixed binary support imposed in this dissertation. One way to achieve that is through the introduction of auxiliary columns. Adding such columns would guarantee that the selected number of columns will always be a fixed number. This generalization would allow us to solve all the applications for B&P presented in Section 1.1.1.

Another research direction is studying the effect of adding cutting planes in the SBP method and this gives rise to Slim Branch and Cut, and Price method. In particular, clique inequalities, wheel inequalities, and other cutting planes can be added to SBP without changing the structure of the pricing problem developed for SBP. Specifically, since the pricing problem in SBP already includes a mechanism to prevent the regeneration of the already generated columns, the implementation of any cutting plane would be straightforward with the only possible change in the set of the prohibited columns.

REFERENCES

[1] V. Acuna, C. E. Ferreira, A. S. Freire, and E. Moreno. Solving the maximum edge biclique packing problem on unbalanced bipartite graphs. *Discrete Applied Mathematics*, 164(1):2–12, 2014.

[2] J. D. Adler and P. B. Mirchandani. The vehicle scheduling problem for fleets with alternative-fuel vehicles. *Transportation Science*, 2016.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Inc., New Jersey, 1993.

[4] J. M. Van Den Akker, J. A. Hoogeveen, and S. L. Van De Velde. Parallel machine scheduling by column generation. *Operations Research*, 47:862–82, 1999.

[5] C. Archetti, N. Bianchessi, and A. Hertz. A branch-and-price algorithm for the robust graph coloring problem. *Discrete Applied Mathematics*, 165:49–59, 2014.

[6] T. P. Bagchi, J. N. D. Gupta, and C. Sriskandarajah. A review of tsp based approaches for flowshop scheduling. *European Journal of Operational Research*, 169(3):816–854, 2006.

[7] R. Baldacci, A. Mingozzi, and R. Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283, 2011.

[8] C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[9] N. Bianchessi, R. Mansini, and M. G. Speranza. The distance constrained multiple vehicle traveling purchaser problem. *European Journal of Operational Research*, 235(1):73–87, 2014.

[10] I. M. Branco and J. D. Coelho. The Hamiltonian p-median problem. *European Journal of Operations Research*, 47(1):86–95, 1990.

[11] J. O. Brunner and J. F. Bard. Flexible weekly tour scheduling for postal service workers using a branch and price. *Journal of Scheduling*, 16(1):129–149, 2013.

[12] R. W. Calvo and R. Cordone. A heuristic approach to the overnight security service problem. *Computers and Operations Research*, 30(9):1269–1287, 2003.

[13] J. Cerdeira. The Hamiltonian k-median problem for any given k is np-complete. Technical Report 14/86, Centro de Estatistica e Aplicacoes, 1986.

[14] S. S. Chawathe. Organizing hot-spot police patrol routes. In *Intelligence and Security Informatics, 2007 IEEE*, pages 79–86, May 2007.

[15] W. H. Cunningham and Y. Wang. Restricted 2-factor polytopes. *Mathematical Programming*, 87(1):87–111, 2000.

[16] M. Desrochers and F. Soumis. A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23(1):1–13, 1989.

[17] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[18] J. Edmonds. Maximum matching and a polyhedron with $0, 1$ vertices. *Journal of Research of the National Bureau of Standards*, 69B(1):125–130, 1965.

[19] H. Glaab and A. Pott. The Hamiltonian p-median problem. *Electronic Journal of Combinatorics*, 7(1):R–42, 2000.

[20] K. M. Glorie, J. J. van de Klundert, and A. P. M. Wagelmans. Kidney exchange with long chains: An efficient pricing algorithm for clearing barter exchanges with branch-and-price. *Manufacturing & Service Operations Management*, 16(4):498–512, 2014.

[21] S. Gollowitzer, L. Gouveia, G. Laporte, D. L. Pereira, and A. Wojciechowski. A comparison of several models for the Hamiltonian p-median problem. *Networks*, 63(4):350–363, 2014.

[22] S. Gualandi, F. Maffioli, and C. Magni. A branch-and-price approach to k-clustering minimum biclique completion problem. *International Transactions in Operational Research*, 20(1):101–117, 2013.

[23] K. Helsgaun. General k-opt submoves for the lin–kernighan tsp heuristic. *Mathematical Programming Computation*, 1(2):119–163, 2009.

[24] F. Hernandez, D. Feillet, R. Giroudeau, and O. Naud. A new exact algorithm to solve the multi-trip vehicle routing problem with time windows and limited duration. *4OR*, 12(3):235–259, 2014.

[25] M. Hewitt, G. Nemhauser, M. Savelsbergh, and J. Song. A branch-and-price guided search approach to maritime inventory routing. *Computers & Operations Research*, 40(5):1410–1419, 2013.

[26] L. Hupp and F. Liers. A polyhedral study of the Hamiltonian p-median problem. *Electronic Notes in Discrete Mathematics*, 41:213–220, 2013.

[27] E. L. Johnson, G. L. Nemhauser, and M. W. P. Savelsbergh. Progress in linear programming-based algorithms for integer programming: An exposition. *INFORMS Journal on Computing*, 12(1):2–23, 2000.

[28] V. Kolmogorov. Blossom v: A new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1):43–67, 2009.

[29] G. Lancia, F. Rinaldi, and P. Serafini. *A Unified Integer Programming Model for Genome Rearrangement Problems*, pages 491–502. Springer International Publishing, Cham, 2015.

[30] G. Laporte, Y. Nobert, and P. Pelletier. Hamiltonian location problems. *European Journal of Operations Research*, 12(1):82–89, 1983.

[31] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

[32] L. A. N. Lorena and E. L. F. Senne. A column generation approach to capacitated p-median problems. *Computers & Operations Research*, 31(6):863–876, 2004.

[33] A. M. Marzouk, E. Moreno-Centeno, and H. Üster. A branch-and-price algorithm for solving the hamiltonian p-median problem. *INFORMS Journal on Computing*, 28(4):674–686, 2016.

[34] A. Mehrotra, E. L. Johnson, and G. L. Nemhauser. An optimization based heuristic for political districting. *Management Science*, 44(8):1100–1114, 1998.

[35] C. Montoya, O. Bellenguez-Morineau, E. Pinson, and D. Rivreau. Branch-and-price approach for the multi-skill project scheduling problem. *Optimization Letters*, 8(5):1721–1734, 2014.

[36] Í. Muter, Ş. Í. Birbil, K. Bülbül, G. Şahin, H. Yenigün, D. Taş, and D. Tüzün. Solving a robust airline crew pairing problem with column generation. *Computers & Operations Research*, 40(3):815 – 830, 2013.

[37] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley&Sons Inc., New York, 1999.

[38] D. Reis, A. Melo, A. L. V. Coelho, and V. Furtado. *Towards Optimal Police Patrol Routes with Genetic Algorithms*, pages 485–491. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[39] A. Rozenknop, R. Wolfler Calvo, L. Alfandari, D. Chemla, and L. Létocart. Solving the electricity production planning problem by a column generation based heuristic. *Journal of Scheduling*, 16(6):585–604, 2013.

[40] D. M. Ryan and B. A. Foster. *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheuling*, chapter An Integer Programming Approach to Scheduling, pages 269–280. North-Holland, Amsterdam, 1981.

[41] F. A. Santos, A. S. da Cunha, and G. R. Mateus. Branch-and-price algorithms for the two-echelon capacitated vehicle routing problem. *Optimization Letters*, 7(7):1537–1547, 2013.

[42] F. A. Santos, G. R. Mateus, and A. S. da Cunha. The pickup and delivery problem with cross-docking. *Computers & Operations Research*, 40(4):1085–1093, 2013.

[43] M. W. P. Savelsberg. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994.

[44] I. Sung and T. Lee. Optimal allocation of emergency medical resources in a mass casualty incident: Patient prioritization by column generation. *European Journal of Operational Research*, 252(2):623–634, 2016.

[45] H. Üster and S.K.S. Kumar. Algorithms for the design of network topologies with balanced disjoint rings. *Journal of Heuristics*, 16(1):37–63, 2008.

[46] F. Vanderbeck. Branching in branch-and-price: a generic scheme. *Mathematical Programming*, 130(2):249–294, 2011.

[47] J. P. Vielma. Mixed integer linear programming formulation techniques. *SIAM Review*, 57(1):3–57, 2015.

[48] M. Wen, S. Ropke, H. L. Petersen, R. Larsen, and O. B. G. Madsen. Full-shipload tramp ship routing and scheduling with variable speeds. *Computers & Operations Research*, 70:1–8, 2016.

[49] M. Williamson, P. Eirinakis, and K. Subramani. Fast algorithms for the undirected negative cost cycle detection problem. *Algorithmica*, 74(1):270–325, 2014.

[50] G. Woumans, L. De Boeck, J. Belien, and S. Creemers. A column generation approach for solving the examination-timetabling problem. *European Journal of Operational Research*, 253(1):178–194, 2016.

APPENDIX A

A.1 UNWCD Based on b-matching Algorithm (Adapted from [49])

Input: $G' = (V, E)$, a weight w_{ij} for all edges $(i, j) \in E$.
Output: A negative cycle or set of node-disjoint negative cycles or conclude that no negative cycle exists.
Step 1: Set $G^* = G'$. Add a zero weight loop (i, i) for each $i \in V(G^*)$ and set $b_i = 2$.
Step 2: Insert two nodes k and l in the middle of each edge $(i, j) \in E(G^*), i \neq j$. Let the weights of edges (i, k) and (l, j) be $w_{ij}/2$ and zero for edge (k, l). Set $b_k = b_l = 1$.
Step 3: Split each node $i \in V(G^*)$ with $b_i = 2$ into i' and i''. While splitting, replace each edge (i, j) by two edges (i', j) and (i'', j) having the same weight as (i, j). Replace loop (i, i) by (i', i'') with same weight.
Step 4: Find a minimum weight perfect matching in G^*, M. If the cost of M is less than zero, go to step 5; otherwise, go to step 6.
Step 5: Conclude that G' has a negative cycle. Use M to return one or more negative cycles and quit.
Step 6: Return that G' has no negative cycle and quit.

The figure below illustrates that if all of the negative cycles on a given graph pass through one node, then UNWCD will only detect and output the most negative among these cycles.

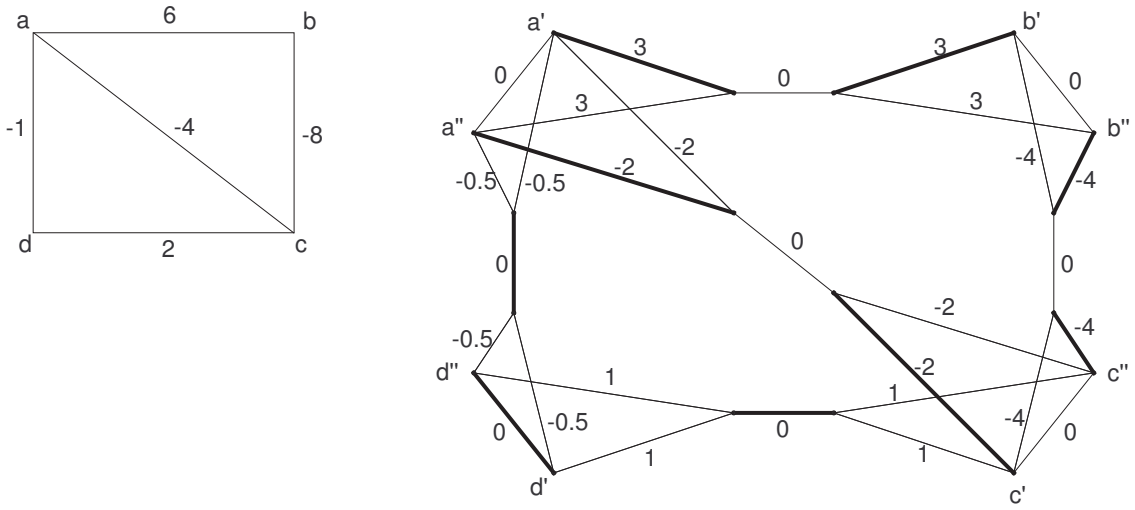


Figure A.1: The original graph G' is on the left (note it has three negative cycles, all passing though node a). The graph G^* is on the right. The bold lines represent the edges in M. The negative cycle detected is a-b-c-a with total cost -6.

A.2 Natural Variable Space Model (NVM) ([21])

We give the natural variable space model as presented in [21]. To this end, some terminology is first defined. Let P be a partition of set of vertices V into m subsets given as $\{S_1, \ldots, S_m\}$ and let $\mathscr{P}_m^l$ be the set of partitions of size m such that the cardinality of each $S_i, i = 1, \ldots, m$ is greater than or equal to l. For a partition P, define E_P as the set of edges straddling pairs of subsets S_v and S_w (i.e., $E_P = \{(i,j) \in E : i \in S_v, j \in S_w; S_v \neq S_w; S_v, S_w \in P\}$), and $\mathscr{C}_p = \{C \subset E : |C| = |V|$ and the edges in C form at most p cycles$\}$.

Now, the mathematical formulation of the natural variable space is given by:

$$\textbf{(NVM)} \quad \text{minimize } H = \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{A.1a}$$

$$\text{subject to} \sum_{\forall(i,j)\in\delta(v)} x_{ij} = 2 \qquad \forall v \in V \tag{A.1b}$$

$$\sum_{(i,j)\in E_P} x_{ij} \geq 2 \qquad \forall P \in \mathscr{P}_{p+1}^3 \tag{A.1c}$$

$$\sum_{(i,j)\notin C} x_{ij} \geq 2 \qquad \forall C \in \mathscr{C}_{p-1} \tag{A.1d}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in E \tag{A.1e}$$

In this model, x_{ij} equals one if edge (i,j) is in the optimal solution, and is zero otherwise. In the constraints (A.1b), $\delta(v)$ denotes the set of edges incident to node $v \in V$, these constraints imply that exactly two of the edges incident to v must be selected. Constraints (A.1c) prevent the formation of more than p cycles; whereas constraints (A.1d) prevent the formation of fewer than p cycles. The separation algorithms for constraints (A.1c) and (A.1d) are provided in [21].

A.3 Detailed Performance Statistics of B&P

Ins.	Stat./p	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
gr17	case 2 %	34	37	58	•									
	case 3 %	0	0	0	•									
	case 4 %	66	63	42	•									
	pricing steps	148	84	36	•									
	no columns	168	111	59	•									
	branching nodes	1	1	1	•									
	Av set Q size	4	5	6	•									
	Av Alg. 3 iter.	3	3	3	•									
gr21	case 2 %	48	52	85	58	66	26							
	case 3 %	17	34	15	34	34	74							
	case 4 %	34	14	0	7	0	0							
	pricing steps	829	184	46	149	41	65							
	no columns	872	226	83	179	67	82							
	branching nodes	31	7	1	7	1	1							
	Av set Q size	5	6	0	6	0	0							
	Av Alg. 3 iter.	2	3	0	2	0	0							
gr24	case 2 %	65	•	76	71	48	40	28						
	case 3 %	0	•	24	29	52	60	72						
	case 4 %	35	•	0	0	0	0	0						
	pricing steps	227	•	63	56	198	111	106						
	no columns	267	•	103	91	222	132	122						
	branching nodes	1	•	1	1	19	9	1						
	Av set Q size	4	•	0	0	0	0	0						
	Av Alg. 3 iter.	2	•	0	0	0	0	0						
fri26	case 2 %	54	59	55	42	67	•	45						
	case 3 %	0	0	7	24	0	•	55						
	case 4 %	46	41	38	34	33	•	0						
	pricing steps	327	111	141	152	58	•	58						
	no columns	383	160	198	208	104	•	90						
	branching nodes	1	1	3	3	1	•	1						
	Av set Q size	5	6	7	7	7	•	0						
	Av Alg. 3 iter.	2	2	2	2	2	•	0						
swiss42	case 2 %	49	34	33	42	45	•	72	85	70	79	52	56	22
	case 3 %	0	3	0	6	8	•	15	15	30	21	48	44	78
	case 4 %	51	63	67	51	47	•	12	0	0	0	0	0	0
	pricing steps	1488	863	735	608	897	•	266	148	81	73	122	79	125
	no columns	1774	1078	905	749	1041	•	364	241	164	154	202	145	172
	branching nodes	1	3*	3	5	9	•	5	3	1	1	3	5	3
	Av set Q size	7	6	5	8	8	•	9	0	0	0	0	0	0
	Av Alg. 3 iter.	2	2	2	2	2	•	2	0	0	0	0	0	0
dantzig42	case 2 %	52	49	64	48	71	27	•	89	50	70	38	48	40
	case 3 %	0	0	4	3	1	7	•	11	50	30	62	52	60
	case 4 %	48	51	31	49	28	66	•	0	0	0	0	0	0
	pricing steps	2148	744	315	510	155	1182	•	63	123	89	130	294	324
	no columns	2529	949	470	667	275	1316	•	166	201	172	209	705	822
	branching nodes	1*	1	5	9	3	32	•	1	7	3	5	49	52*
	Av set Q size	6	8	8	7	9	8	•	0	0	0	0	0	0
	Av Alg. 3 iter.	2	3	2	2	2	2	•	0	0	0	0	0	0

Table A.1: Performance Statistics of B&P for graphs with 17 to 42 nodes.
(*) represents the instances that are not solved to optimality.

Ins.	Stat./p	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
gr48	case 2 %	36	58	55	47	•	52	36	30	37	41	57	51	52	21	10			
	case 3 %	0	0	0	16	•	47	64	70	63	59	43	49	48	79	90			
	case 4 %	64	42	45	36	•	1	0	0	0	0	0	0	0	0	0			
	pricing steps	11	792	453	954	•	367	512	572	240	355	143	85	102	209	412			
	no columns	16	1110	684	1178	•	520	611	646	336	473	230	177	196	259	396			
	branching nodes	1*	1*	2*	29	•	21	45	47	15	19	3	1	1	23	55			
	Av set Q size	66	7	6	6	•	5	0	0	0	0	0	0	0	0	0			
	Av Alg. 3 iter.	22	3	3	2	•	2	0	0	0	0	0	0	0	0	0			
hk48	case 2 %	72	51	40	62	•	75	84	60	55	40	41	46	59	59	10			
	case 3 %	0	0	2	0	•	25	16	40	45	60	59	54	41	41	90			
	case 4 %	28	49	59	38	•	0	0	0	0	0	0	0	0	0	0			
	pricing steps	18	746	818	290	•	203	92	281	101	300	220	114	94	94	392			
	no columns	28	1028	1104	474	•	379	227	432	216	423	335	207	208	197	468			
	branching nodes	1*	1	7	1	•	5	1	15	1	15	11	7	1	1	53			
	Av set Q size	62	7	7	8	•	11	0	0	0	0	0	0	0	0	0			
	Av Alg. 3 iter.	21	2	2	2	•	2	0	0	0	0	0	0	0	0	0			
eil51	case 2 %	64	•	57	62	72	59	57	57	45	58	49	52	71	64	39	11		
	case 3 %	0	•	9	22	28	41	43	43	55	42	51	48	29	36	61	89		
	case 4 %	36	•	34	16	0	0	0	0	0	0	0	0	0	0	0	0		
	pricing steps	2066	•	2570	2362	674	541	1424	358	871	204	831	345	90	108	150	509		
	no columns	2414	•	2887	2621	854	685	1588	468	991	323	934	448	204	212	227	564		
	branching nodes	1*	•	31*	51	15	15	50	11	41	5	35	9	1	2	4	35		
	Av set Q size	6	•	6	7	0	0	4	0	0	0	0	0	0	0	0	0		
	Av Alg. 3 iter.	2	•	2	2	0	0	2	0	0	0	0	0	0	0	0	0		
berlin52	case 2 %	46	44	46	35	40	•	81	60	75	72	60	44	37	36	27	19		
	case 3 %	0	0	2	4	4	•	19	35	25	28	41	56	63	64	73	81		
	case 4 %	54	56	53	61	55	•	0	5	0	0	0	0	0	0	0	0		
	pricing steps	1260	1387	1032	1232	623	•	186	575	174	61	200	105	168	162	258	351		
	no columns	1446	1813	1403	1499	865	•	385	752	324	180	335	225	272	268	329	368		
	branching nodes	1*	1*	4*	11	9	•	5	17	5	1	9	7	11	11	43	67		
	Av set Q size	7	7	9	9	10	•	0	12	0	0	0	0	0	0	0	0		
	Av Alg. 3 iter.	2	2	2	2	2	•	0	3	0	0	0	0	0	0	0	0		
brazil58	case 2 %	38	36	54	60	52	65	68	74	40	76	•	73	60	42	25	16	24	17
	case 3 %	0	0	0	0	0	0	0	0	0	0	•	28	40	58	75	84	76	83
	case 4 %	62	64	46	40	48	35	32	26	60	24	•	0	0	0	0	0	0	0
	pricing steps	21	105	832	728	456	343	221	138	169	82	•	40	53	88	151	362	281	268
	no columns	26	130	1257	1087	721	598	435	327	368	261	•	175	178	204	254	422	336	300
	branching nodes	1*	1*	1*	1*	1*	1*	1*	1	5*	1	•	1	1	7	13	63	39	27
	Av set Q size	41	8	9	10	10	11	13	14	9	16	•	0	0	0	0	0	0	0
	Av Alg. 3 iter.	14	3	2	2	2	2	3	3	2	3	•	0	0	0	0	0	0	0

Table A.2: Performance Statistics of B&P for graphs with 48 to 58 nodes.
(*) represents the instances that are not solved to optimality.

Ins.	Stat./p	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
st70	case 2 %	55	71	44	40	37	52	54	48	61	80	61	•	98	53	95	68	69	45	50
	case 3 %	0	0	0	0	0	0	0	0	0	0	0	•	1	21	5	32	31	55	50
	case 4 %	45	29	56	60	63	48	46	52	39	20	39	•	1	26	0	0	0	0	0
	pricing steps	20	7	1148	870	679	376	287	271	228	128	145	•	91	317	56	62	105	167	152
	no columns	26	15	1748	1434	1140	743	630	572	490	374	384	•	322	566	256	234	289	340	320
	branching nodes	1*	1*	1*	1*	1*	1*	1*	1*	1	1	1	•	1	11	1	3	3	11	9
	Av set Q size	55	122	14	15	16	16	17	14	13	19	15	•	18	16	0	0	0	0	0
	Av Alg. 3 iter.	19	31	2	2	2	2	2	3	3	3	2	•	2	3	0	0	0	0	0
eil76	case 2 %	50	•	81	81	84	84	84	79	87	85	87	62	73	59	47	55	41	55	54
	case 3 %	0	•	1	18	16	16	16	21	13	15	13	38	27	41	53	45	59	45	46
	case 4 %	50	•	19	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	pricing steps	1125	•	1515	1284	699	904	961	369	324	262	425	598	324	408	554	251	438	335	384
	no columns	1320	•	2143	1870	1162	1322	1365	655	634	546	714	851	569	628	764	437	634	545	604
	branching nodes	1*	•	1*	7	3	8	6*	1	2	3	4	11	5	7	21	3	15	9	11
	Av set Q size	7	•	10	9	7	0	2	0	0	0	0	0	0	0	0	0	0	0	0
	Av Alg. 3 iter.	3	•	2	2	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0
pr76	case 2 %	38	56	50	51	43	38	•	53	52	50	47	45	59	57	42	62	42	47	66
	case 3 %	0	0	0	0	0	0	•	27	29	49	52	55	41	39	58	38	58	53	34
	case 4 %	62	44	50	49	57	62	•	20	19	1	0	0	0	4	0	0	0	0	0
	pricing steps	680	1527	1234	1251	1011	938	•	2386	2594	4991	4780	4593	4987	4189	621	536	333	273	110
	no columns	783	2130	1715	1740	1444	1307	•	2758	2984	5255	4992	4743	5139	4363	804	729	497	453	270
	branching nodes	1*	1*	1*	1*	2*	2*	•	26*	32*	94	106*	109*	117*	92*	7	11	5	9	1
	Av set Q size	6	13	14	13	15	15	•	15	15	15	18	0	0	15	0	0	0	0	0
	Av Alg. 3 iter.	2	2	2	2	2	2	•	2	2	2	2	0	0	2	0	0	0	0	0
gr96	case 2 %	36	27	60	34	52	47	50	47	•	56	63	87	70	56	57	68	72	70	61
	case 3 %	0	0	0	0	0	0	0	0	•	7	11	12	30	43	38	31	28	30	39
	case 4 %	64	73	40	66	48	53	50	53	•	37	26	2	0	2	5	0	0	0	0
	pricing steps	275	64	1039	79	1094	1014	1040	975	•	1340	1684	434	463	2670	2527	498	370	512	405
	no columns	295	77	1845	103	1834	1680	1673	1537	•	1961	2389	856	819	3147	2993	820	668	823	676
	branching nodes	1*	1*	1*	1*	1*	1*	1*	2*	•	4*	10*	2*	5	59*	45*	4*	5*	9	5
	Av set Q size	6	10	14	11	18	19	20	20	•	21	20	18	0	17	23	8	10	0	0
	Av Alg. 3 iter.	2	3	2	3	2	2	2	2	•	2	2	2	0	3	3	2	2	0	0
rat99	case 2 %	42	56	70	•	75	89	96	93	99	92	57	37	54	38	43	39	31	42	40
	case 3 %	0	0	0	•	0	3	3	3	0	7	42	63	46	62	57	61	69	58	60
	case 4 %	58	44	30	•	25	9	1	4	1	2	0	0	0	0	0	0	0	0	0
	pricing steps	57	39	1453	•	980	547	477	405	361	237	533	779	537	1229	738	804	1011	589	788
	no columns	68	51	2455	•	1864	1305	1117	1022	899	721	1061	1189	1007	1655	1104	1123	1337	872	1069
	branching nodes	1*	1*	1*	•	1*	1*	1	1	1	1	14	29	13	47	27	41	45	19	27
	Av set Q size	14	20	12	•	11	11	11	2	7	5	18	0	2	0	0	5	5	0	0
	Av Alg. 3 iter.	4	4	2	•	2	2	3	1	3	2	3	0	1	0	0	2	3	0	0
rd100	case 2 %	43	33	62	58	54	47	40	39	38	45	45	58	•	71	92	87	73	57	62
	case 3 %	0	0	0	0	0	0	0	0	0	0	0	0	•	1	4	13	27	42	34
	case 4 %	57	67	38	42	46	53	60	61	62	55	55	42	•	28	4	0	0	1	5
	pricing steps	7	18	1189	1065	991	823	745	717	659	571	536	457	•	588	298	345	658	2688	880
	no columns	10	26	2163	2029	1845	1551	1412	1337	1211	1078	1027	903	•	1073	741	770	1063	3129	1263
	branching nodes	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1	1	•	2	1	3	9	52	14*
	Av set Q size	106	33	17	18	19	20	21	23	22	21	19	19	•	23	21	0	0	24	24
	Av Alg. 3 iter.	34	9	2	2	2	2	2	2	2	2	2	2	•	2	2	0	0	2	3

Table A.3: Performance Statistics of B&P for graphs with 70 to 100 nodes for $p \leq 20$.
(*) represents the instances that are not solved to optimality.

Ins.	Stat./p	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**
	case 2 %	95	68	69	45	50								
	case 3 %	5	32	31	55	50								
	case 4 %	0	0	0	0	0								
st70	pricing steps	56	62	105	167	152								
	no columns	256	234	289	340	320								
	branching nodes	1	3	3	11	9								
	Av set Q size	0	0	0	0	0								
	Av Alg. 3 iter.	0	0	0	0	0								
	case 2 %	47	55	41	55	54								
	case 3 %	53	45	59	45	46								
	case 4 %	0	0	0	0	0								
eil76	pricing steps	554	251	438	335	384								
	no columns	764	437	634	545	604								
	branching nodes	21	3	15	9	11								
	Av set Q size	0	0	0	0	0								
	Av Alg. 3 iter.	0	0	0	0	0								
	case 2 %	42	62	42	47	66								
	case 3 %	58	38	58	53	34								
	case 4 %	0	0	0	0	0								
pr76	pricing steps	621	536	333	273	110								
	no columns	804	729	497	453	270								
	branching nodes	7	11	5	9	1								
	Av set Q size	0	0	0	0	0								
	Av Alg. 3 iter.	0	0	0	0	0								
	case 2 %	57	68	72	70	61	40	30	21	18	16	18	11	
	case 3 %	38	31	28	30	39	60	70	79	82	84	82	89	
	case 4 %	5	0	0	0	0	0	0	0	0	0	0	0	
gr96	pricing steps	2527	498	370	512	405	630	512	504	499	415	401	509	
	no columns	2993	820	668	823	676	848	678	604	584	481	489	491	
	branching nodes	45*	4*	5*	9	5	31	31	92*	96*	93*	84*	70*	
	Av set Q size	23	8	10	0	0	0	0	0	0	0	0	0	
	Av Alg. 3 iter.	3	2	2	0	0	0	0	0	0	0	0	0	
	case 2 %	43	39	31	42	40	47	41	40	30	33	38	20	12
	case 3 %	57	61	69	58	60	53	59	60	70	68	62	80	88
	case 4 %	0	0	0	0	0	0	0	0	0	0	0	0	0
rat99	pricing steps	738	804	1011	589	788	291	212	272	346	320	239	370	544
	no columns	1104	1123	1337	872	1069	514	431	485	545	516	445	480	536
	branching nodes	27	41	45	19	27	7	7	8	23	19	17	90	71*
	Av set Q size	0	5	5	0	0	0	0	0	0	0	0	0	0
	Av Alg. 3 iter.	0	2	3	0	0	0	0	0	0	0	0	0	0
	case 2 %	92	87	73	57	62	46	52	39	34	32	22	21	14
	case 3 %	4	13	27	42	34	54	48	61	66	68	78	79	86
	case 4 %	4	0	0	1	5	0	0	0	0	0	0	0	0
rd100	pricing steps	298	345	658	2688	880	318	321	310	276	192	435	486	548
	no columns	741	770	1063	3129	1263	627	633	579	531	431	635	657	626
	branching nodes	1	3	9	52	14*	15	17	21	19	11	55*	71*	72*
	Av set Q size	21	0	0	24	24	0	0	0	0	0	0	0	0
	Av Alg. 3 iter.	2	0	0	2	3	0	0	0	0	0	0	0	0

Table A.4: Performance Statistics of B&P for graphs with 70 to 100 nodes for $p > 20$.
(*) represents the instances that are not solved to optimality.

Ins.	Stat./p	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
	case 2 %	38	59	43	32	52	51	50	48	24	52	48	•	65	62	68	75	66	51	43
	case 3 %	0	0	0	0	0	0	0	0	0	0	0	•	7	2	31	25	33	49	57
	case 4 %	63	41	57	68	48	49	50	52	76	48	52	•	28	35	1	1	0	0	0
kroa100	pricing steps	8	27	7	60	925	898	959	824	800	756	775	•	404	405	187	166	252	325	262
	no columns	12	35	16	86	1770	1740	1812	1599	1554	1418	1533	•	890	912	567	509	632	679	585
	branching nodes	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	•	3*	2*	3*	3	7	13	15
	Av set Q size	52	32	73	15	18	17	19	19	13	19	19	•	21	19	8	20	4	0	0
	Av Alg. 3 iter.	14	8	14	3	2	2	2	2	2	2	2	•	3	2	2	2	2	0	0
	case 2 %	65	34	23	51	51	54	52	60	63	73	74	88	80	92	74	85	72	80	•
	case 3 %	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	•
	case 4 %	35	66	77	49	49	46	48	40	37	27	26	12	20	8	26	15	28	20	•
krob100	pricing steps	17	65	74	988	944	939	811	623	646	397	302	242	291	203	250	241	239	192	•
	no columns	21	86	84	1954	1864	1847	1594	1289	1320	965	794	649	719	574	648	639	617	548	•
	branching nodes	1*	1*	1*	1*	1*	1*	1*	1*	2*	1*	1*	1*	1*	1*	2*	1*	1*	2*	•
	Av set Q size	66	14	10	16	17	20	21	20	17	20	18	13	17	15	16	16	26	22	•
	Av Alg. 3 iter.	22	4	3	2	2	2	2	2	2	3	2	2	2	2	3	4	3	4	•
	case 2 %	50	83	36	62	51	61	60	56	67	83	91	91	•	97	98	74	76	44	44
	case 3 %	0	0	0	0	0	0	0	0	0	0	0	0	•	2	1	26	24	56	56
	case 4 %	50	17	64	38	49	39	40	44	33	17	8	9	•	1	1	0	0	0	0
krod100	pricing steps	12	6	28	783	99	880	689	703	483	307	219	231	•	118	107	113	107	346	344
	no columns	18	15	42	1709	145	1802	1578	1554	1184	918	747	715	•	529	512	486	463	726	691
	branching nodes	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	•	1*	1*	1	1	21	26
	Av set Q size	114	255	20	11	18	16	18	18	17	17	5	14	•	4	19	0	0	0	18
	Av Alg. 3 iter.	38	51	4	2	3	2	2	2	2	2	2	3	•	2	4	0	0	0	4
	case 2 %	34	38	39	40	49	58	61	54	57	65	56	56	57	74	61	64	88	77	•
	case 3 %	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	•
	case 4 %	66	63	61	60	51	42	39	46	43	35	44	44	43	26	39	36	12	10	•
lin105	pricing steps	134	32	497	42	297	838	664	724	570	469	426	303	287	209	341	215	83	272	•
	no columns	142	42	608	62	469	1811	1484	1620	1314	1191	1020	813	773	698	847	639	423	714	•
	branching nodes	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	2*	8*	5*	1*	9*	•
	Av set Q size	7	24	10	22	12	19	21	21	21	20	21	19	20	26	10	13	37	35	•
	Av Alg. 3 iter.	3	6	2	4	2	2	2	2	2	2	2	2	2	4	3	4	3	6	•
	case 2 %	63	30	83	86	75	70	71	65	66	73	72	81	86	•	79	77	66	81	84
	case 3 %	0	0	0	0	0	0	0	0	0	0	0	0	1	•	19	22	34	19	16
	case 4 %	38	70	17	14	25	30	28	35	34	27	28	19	13	•	2	0	0	0	0
gr120	pricing steps	8	30	6	948	1205	998	892	927	882	774	861	736	665	•	1318	2098	3014	1329	2923
	no columns	14	39	17	2071	2447	2134	2039	1946	1874	1636	1707	1502	1386	•	2116	3031	3910	2125	3934
	branching nodes	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	•	5*	13*	29*	6	13
	Av set Q size	88	23	243	15	16	18	19	19	20	20	20	20	20	•	26	5	0	5	0
	Av Alg. 3 iter.	24	5	48	2	2	2	2	2	2	2	2	2	2	•	2	2	0	2	0
	case 2 %	31	54	63	76	79	76	65	61	69	74	•	74	71	87	97	82	77	67	56
	case 3 %	0	0	0	0	0	0	0	0	0	0	•	1	2	8	3	18	23	33	44
	case 4 %	69	46	38	24	21	24	35	39	31	26	•	25	27	4	1	0	0	0	0
bier127	pricing steps	245	384	8	1602	1103	968	1009	949	1147	1229	•	1263	1060	359	352	506	324	367	484
	no columns	277	425	18	2936	2208	2080	2120	2048	2355	2345	•	2348	2094	1056	1078	1263	921	894	1035
	branching nodes	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	•	2*	2*	2*	1	5	3	5	7
	Av set Q size	10	9	91	18	18	19	23	22	22	23	•	25	25	22	9	15	0	0	0
	Av Alg. 3 iter.	4	2	17	2	2	2	2	2	2	2	•	2	2	2	1	2	0	0	0

Table A.5: Performance Statistics of B&P for graphs with 100 to 127 nodes for $p \leq 20$.
(*) represents the instances that are not solved to optimality.

Ins.	Stat./p	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**
	case 2 %	42	34	33	34	45	34	22	18	22	19	22	19	11							
	case 3 %	58	66	67	66	55	66	78	82	78	81	78	81	89							
	case 4 %	0	0	0	0	0	0	0	0	0	0	0	0	0							
kroa100	pricing steps	175	487	632	308	116	210	465	524	481	411	418	352	471							
	no columns	461	771	899	555	344	458	646	698	660	572	568	478	541							
	branching nodes	11	35	53	29	5	13	61	73	67	83	79*	89*	84*							
	Av set Q size	0	0	0	0	0	0	0	0	0	0	0	0	0							
	Av Alg. 3 iter.	0	0	0	0	0	0	0	0	0	0	0	0	0							
	case 2 %	97	78	97	78	55	64	38	41	66	40	17	19	13							
	case 3 %	1	11	1	22	45	35	62	59	34	60	83	81	87							
	case 4 %	2	12	2	0	0	1	0	0	0	0	0	0	0							
krob100	pricing steps	122	276	92	83	125	165	172	76	62	75	323	365	443							
	no columns	468	663	438	396	431	503	435	309	297	298	484	494	521							
	branching nodes	1*	3*	1*	1	3	4*	5	1	1	3	47	91	80*							
	Av set Q size	5	25	5	0	0	4	0	0	0	0	0	0	0							
	Av Alg. 3 iter.	3	4	2	0	0	2	0	0	0	0	0	0	0							
	case 2 %	34	42	44	51	48	48	29	24	31	28	19	20	16							
	case 3 %	66	58	56	49	52	52	71	76	69	72	81	80	84							
	case 4 %	0	0	0	0	0	0	0	0	0	0	0	0	0							
krod100	pricing steps	492	351	516	266	188	127	256	505	296	366	460	457	441							
	no columns	808	684	874	574	475	407	517	737	527	618	650	639	560							
	branching nodes	35	21	23	9	7	7	15	45	23	33	80	81*	86*							
	Av set Q size	0	0	0	0	0	0	0	0	0	0	0	0	0							
	Av Alg. 3 iter.	0	0	0	0	0	0	0	0	0	0	0	0	0							
	case 2 %	66	61	57	63	46	58	73	30	34	31	28	23	23	16	8					
	case 3 %	34	39	43	34	51	42	27	70	66	69	72	77	77	84	92					
	case 4 %	0	0	0	3	4	0	0	0	0	0	0	0	0	0	0					
lin105	pricing steps	300	305	177	229	598	165	94	555	358	517	300	421	432	500	631					
	no columns	717	699	505	577	967	496	390	819	622	767	554	628	599	620	677					
	branching nodes	11	11	7	9	52*	5	1	69	33	66*	29	64*	65*	61*	52*					
	Av set Q size	0	0	0	37	23	0	0	0	0	0	0	0	0	0	0					
	Av Alg. 3 iter.	0	0	0	6	3	0	0	0	0	0	0	0	0	0	0					
	case 2 %	84	82	78	70	74	66	46	43	43	61	39	49	45	73	67	33	24	28	22	
	case 3 %	16	18	22	30	26	34	54	57	57	39	61	51	55	27	33	67	76	72	78	
	case 4 %	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
gr120	pricing steps	359	428	573	2957	409	303	819	1363	1000	592	688	696	566	223	165	447	351	418	445	
	no columns	861	846	1051	3705	830	666	1254	1829	1425	1009	1027	1163	952	609	519	785	629	710	643	
	branching nodes	1	1	3	27	1	1	17	35	23	9	15	15	11	1	1	33	35*	45*	43*	
	Av set Q size	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	Av Alg. 3 iter.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	case 2 %	71	60	75	73	65	55	67	64	55	29	34	35	27	23	23	21	24	22	19	19
	case 3 %	29	40	25	27	35	45	33	36	45	71	66	65	73	77	77	78	76	78	81	81
	case 4 %	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bier127	pricing steps	286	465	263	207	275	241	166	140	183	468	447	298	393	364	320	467	412	405	406	554
	no columns	850	1026	759	691	716	650	598	513	593	797	806	634	692	621	565	730	682	666	635	784
	branching nodes	3	9	1	1	3	1	1	1	5	31	19	9	37*	40*	37*	39*	42*	37*	37*	39*
	Av set Q size	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0
	Av Alg. 3 iter.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0

Table A.6: Performance Statistics of B&P for graphs with 100 to 127 nodes for $p > 20$.
(*) represents the instances that are not solved to optimality.