ABSTRACT

PEKER CANSIZOĞLU, AYŞEGÜL SELCAN. Solving the Multi-dimensional 0-1 Knapsack Problem using Depth-k Canonical Cuts. (Under the direction of Dr. Thom J. Hodgson.)

Many decision problems can be formulated as a Multi-dimensional 0-1 Knapsack Problem (MKP). These problems are well studied in the literature using different methods, such as cutting plane algorithms, heuristics and branching techniques. The purpose in this study is to find geometric insights and implement these insights to develop efficient techniques for solving the MKP, and to test the developed techniques on various test suites.

First, we develop an efficient cutting plane algorithm with canonical cuts for solving the MKP. We improve this algorithm adding features like preprocessing, n-way branching, and imposing a complexity limit on canonical cuts. We also investigate the potential of this algorithm to be used as a heuristic with early termination. Preliminary experimental results show that these heuristics have good potential.

Second, we extend this study by focusing on canonical cuts. We introduce the depth concept for canonical cuts. In literature, depth-1 canonical cuts are also referred to as clauses in constraint programming or closure cuts in integer programming. Since the practicality of these cuts is not validated, we extend the canonical cut theory and use these new results to improve practicality of the canonical cuts. We introduce depth-k canonical cut. We propose three different implementations of canonical cut to increase their practicality. We integrate depth-1 and depth-2 canonical cuts within the SAS/OR MILP solver and evaluate performance improvement over randomly generated correlated test problems in addition to OR Library test problems. Overall, new implementations of canonical cuts are shown to be very effective over these test suites.

In the last part of the study we focus on the Winner Determination Problem (WDP) for Multi-Resource Combinatorial Auctions which can be formulated as an MKP. These problems are in the difficult MKP category. We test the effectiveness of different implementations of the canonical cuts over a test suite of these problems. These results show that there is a significant performance increase with canonical cuts.

Solving the Multi-dimensional 0-1 Knapsack Problem using Depth-k Canonical Cuts

by
Ayşegül Selcan Peker Cansızoğlu

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Industrial Engineering

Raleigh, North Carolina

2011

APPROVED BY:

Dr. Shu-Cherng Fang Co-chair of Advisory Committee	Dr. Russell E. King
Dr. Michael G. Kay	Dr. Donald E. Bitzer

Dr. Thom J. Hodgson
Co-chair of Advisory Committee

DEDICATION

To Omer, Kerem and My Family

BIOGRAPHY

Ayşegül Selcan Peker Cansızoğlu received her B.S. in Industrial Engineering from Bilkent University in 2003. She was accepted to the Ph. D. program at North Carolina State University right after receiving her B. S. degree. She focused on mathematical programming during her graduate study. In 2008, she started working at SAS Institute's Advanced Analytics R&D Division as an Analytical Software Tester for SAS/OR. In 2011, she joined the OR Center of Excellence as an Operations Research Specialist at SAS Institute. Her research interests include integer optimization and large-scale non-linear optimization.

ACKNOWLEDGEMENTS

My sincere thanks go to my advisors, Dr. Thom J. Hodgson and Dr. Shu-Cherng Fang, and to the other members of my Ph.D. committee at North Carolina State University (NCSU), for their invaluable help and support during the entire research.

This thesis would not have been possible without support and encouragement I received from my manager, Ivan Oliveira at SAS Institute. In particular, I would like to thank Rob Pratt for his continuous support and his great advice, Amar Narisetty for his invaluable help, Joshua Griffin for his great pep-talks. I appreciate the unconditional support I received from my colleague Lindsey Puryear. I also thank my colleagues Philipp Christophel, Gehan Corea, Matthew Galati, Mustafa Onur Kabul, Emily Lada, Yu-min Lin, Michelle Opp, Imre Polik, Yan Xu and Wenwen Zhou for their support.

I acknowledge my friends, Merivan Tunc Reis, Engin Reis and Emine Yaylali who constantly supported me during my study.

Lastly and most importantly, I deeply thank my dear son Kerem who fills my life with joy, my husband Omer Cansizoglu, my parents Hacer and Mehmet Peker, my brother Cagri Peker and my sister Hilal Peker for their unconditional support and sacrifice.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

General Information and Problem Definition

Integer programming is a mathematical tool for solving optimization problems. The decision is to determine the values of variables so that an objective function is optimized within the constraints of the problem. At least one of the variable values is picked from an integer domain. There is a special class of integer programming problems which restricts the variables to be either 0 or 1 and constraints to be linear. This type of problems are called $0-1$ or binary problems. In this study we focus on the Multi-dimensional 0-1 Knapsack Problem, which is a special case of binary problems.

The following formulation defines a mixed-integer linear programming (MILP) problem:

$$\begin{aligned} \max\ & c^x x + c^y y \\ & A^x x + A^y y \leq b \\ & x, y \geq 0 \\ & x \text{ integer} \end{aligned}$$

where $x = (x_1, \ldots, x_n)^T$ and $y = (y_1, \ldots, y_p)^T$ are the decision variables, c^x and c^y are the objective coefficient vectors for x and y with sizes $n \times 1$ and $p \times 1$, A^x and A^y are the constraint coefficient matrices for x and y with sizes $m \times n$ and $m \times p$, and b is the right hand side vector with size $m \times 1$. A pure integer programming problem (IP) is a special case of MILP that

requires all the variables to be integer as given below:

$$\begin{aligned} \max\ & c^x x \\ & A^x x \leq b \\ & x \geq 0 \\ & x \text{ integer} \end{aligned}$$

A binary integer programming problem (BIP) is an IP with all variables restricted to be either 0 or 1 as given below:

$$\begin{aligned} \max\ & c^x x \\ & A^x x \leq b \\ & x \in \{0,1\} \end{aligned}$$

In this study we focus on solving the Multi-dimensional 0-1 Knapsack Problem (MKP) which is an important problem in both applied and theoretical operations research. MKP is a special case of general 0-1 integer problems. The problem is to find the most desirable set of items given a quantitative measure of desirability for each item, weight for each item, and the maximum total weight the knapsack can carry. It is possible to add more limitations such as maximum total volume, maximum total cost, etc. Adding more limitations to the problem makes it multi-dimensional. The mathematical formulation of MKP is similar to IP. For MKP, we give the following formulation, which has objective vector and constraint matrix extended to describe the problem better:

$$\begin{aligned} \max \quad & f = \textstyle\sum_{j=1}^{n} c_j x_j \\ \text{s.t.} \quad & \textstyle\sum_{j=1}^{n} a_{ij} \leq b_i, i \in M = \{1,\ldots,m\} \\ & x_j \in \{0,1\}, j \in N = \{1,\ldots,n\}. \end{aligned} \tag{1.1}$$

In this formulation n is the number of variables and m is the number of knapsack constraints. The capacity of each constraint is b_i for $i \in \{1,\ldots,m\}$. Each item consumes a_{ij} of the resource i. The contribution of each item to the objective function is c_j for $j \in \{1,\ldots,n\}$. The goal of this problem is finding the subset of items that gives the best objective value without exceeding the limits of resources. All the coefficients are nonnegative.

In literature, it has been shown that existence of special constraints or sparsity helps in developing efficient algorithms to solve problems [3]. Although MKP is a special case of 0-1 integer problems, it does not have special structure constraints and it has dense structure. This makes some instances of MKP hard to solve.

1.1 General Integer Programming Solving Methodologies

This section describes the following three general integer programming solving methodologies.

- Branch and Bound (B&B)
- Cutting Plane Method
- Branch and Cut

1.1.1 Branch and Bound

A branch and bound (B&B) algorithm was introduced by Land and Doig [33]. It is an implicit enumeration of the feasible region of the integer problem. This algorithm requires solving a relaxation of the integer problem. A widely used relaxation is LP relaxation, in which the integrality requirement on the variables is relaxed. If an optimal solution of the relaxation is an integer solution, this solution is also an optimal solution of the integer problem. Infeasibility of the relaxation problem implies infeasibility of the original problem. Each node represents a point at which the relaxation problem is solved. Nodes have child nodes which are the end points of a branching. One widely used strategy is branching on each integer variable x_i given that it has fractional value x_i^* in the solution of relaxation. There are two child nodes in this branching. One of them is $x_i \leq \lfloor x_i^* \rfloor$ in addition to the relaxation problem of the parent node. The other child node is $x_i \geq \lceil x_i^* \rceil$ in addition to the relaxation problem of the parent node. If the relaxation at a node has an integer optimal solution, then that node is fathomed. This solution provides a bound for the rest of the search. If an optimal solution of relaxation is not as good as the bound, this node is also fathomed. Convergence happens when all nodes are fathomed. A common disadvantage of this method is that the branching tree might grow rapidly and become harder to keep track of.

1.1.2 Cutting Plane Method

Gomory [28] introduced the first cutting plane methods in 1958. The basic idea of these methods is to find a valid inequality such that feasible solutions of the LP relaxation are cut without cutting any feasible integer solution. For every MILP, there exists a linear program (LP) such that all the extreme points of the LP are feasible to the MILP. The aim of cutting plane methodologies is to force the LP relaxation of MILP to get closer to the convex hull of the feasible integer solutions. These approaches are iterative. Cuts are added until a sufficient condition for termination is satisfied. Gomory [27] showed that cutting plane algorithms converge in a finite number of iterations. When cutting planes methodologies are used as part of branch-and-bound algorithms, they can significantly improve the performance.

Cuts are categorized as generic cuts and structure cuts. Generic cuts are applicable to any MILP problem. Gomory fractional cuts are an example for this category [28]. Structure cuts require the problem to have a special form. Flow cover inequalities are in this category [44].

1.1.3 Branch and Cut

Branch and Cut is currently the most widely used methodology for solving MILPs. This is a hybrid method that is based on using cuts at the nodes of the branching tree. When the relaxation is solved and no sufficient condition for fathoming is satisfied, cuts are added. Then the relaxation is solved again. The main idea is to improve the formulation of the problem as much as possible at every node to decrease the branching. Having strong cuts is crucial for the efficiency of this method. In literature there are surveys on branch and cut algorithms such as [42].

1.2 Applications of MKP

There are many applications of MKP and some of them are studied in academic literature. The initial application of this problem is a capital budgeting problem studied by Lorie and Savage [37]. Nemhauser and Ullman [43] define a capital budgeting problem with expenditure limitations as a 0-1 knapsack problem. Allocation of databases in a distributed computer system is another application [23]. The motivation for our study is to improve the performance of the solvers for the multi-resource combinatorial auction problem. The winner determination problem for the multi-resource combinatorial auction problem can be formulated as an MKP. This problem is in the category of difficult MKP. Literature information on this application is given in later sections. Many other applications have MKP as a subproblem, such as multicommodity network optimization problems [22].

1.3 Literature Review

Solution methodologies are categorized into exact and heuristic methods. In this study we propose an exact algorithm so we focus more on the studies of exact algorithms to solve MKP. The first exact algorithms were developed in the 1960s. Gilmore and Gomory [25] suggested a dynamic programming (DP) approach to solve this problem. Weingartner and Ness [62] integrated heuristics into the DP framework. Marsten and Morin [40] used dynamic programming, branch and bound LP bounds and heuristics to effectively solve MKP. These studies used recursive relationships to improve the branch and bound approach for solving MKP [56]. Soyster [55] used implicit enumeration to solve a subproblem of MKP and add a disjunctive cut. Soyster's work has some similarities with our work. Implicit enumeration based techniques

were first investigated by Balas [5] and Glover [26]. Shih [53] improved the effectiveness of LP based B&B using the special structure of MKP. In this study, branching rules and bounds are provided by LP relaxations of m single constrained knapsack problems. However, he could not avoid the excessive space requirement and this method did not perform very well when resource constraints are tight. Later, academics focused more on using Lagrangian relaxations to provide better bounds for this problem. Greenberg and Pierskalla [29] showed that the bound can not be improved more than the maximum objective function coefficient if Lagrangian relaxation is used instead of LP relaxation. More study has been done on Lagrangian relaxation as the bound while solving MKP, however it is not shown to be much more effective than LP relaxation for MKP [15, 31, 8, 9].

Academics also studied surrogate relaxations of MKP. Gavish and Pirkul [24] embedded approximate algorithms into B&B to get surrogate bounds and some implications to reduce problem size. Freville and Plateau [20] also used surrogate relaxations. The performance did not improve, however results were comparable to [24]. Solving MKP problems to optimality might not be necessary or efficient for some problems. Wymann [63] studied how to determine whether to use an exact or an heuristic approach.

Recent works on the multi-dimensional knapsack problem are mostly focused on heuristics. Fleszar and Hindi [17] proposed a fast and effective heuristic. Their methods are based on solving the LP relaxation of the problem and post-processing to increase the quality of the solutions. They used reduced costs to fix the variables at certain values and repeated this procedure until all variables were fixed.

Bertsimas and Demir [10] proposed a dynamic programming based heuristic. In this approach, the LP relaxation is solved. Variables at integer values are fixed at these values in addition to the variables that are close to integer values within a tolerance. The LP relaxation is repeatedly solved until all variables are fixed. The results of these heuristics perform better than the existing heuristics such as Senju and Toyoda's dual gradient heuristic or Loulou and Michaelides's greedy like heuristic [52, 38].

Hanafi and Glover [30] improved surrogate constraints and offered exploitation of nested inequalities. However they have not reported any numerical results.

Akcay pṗroposed a greedy heuristic [2]. This method is intended for general MKP for which variables are in nonnegative integer domain; however, in this study, the authors showed the effectiveness of the heuristics on the 0-1 MKP. In this heuristic, different from other approaches, effective capacity is used to decide to add an item to the knapsack or not. Effective capacity is the maximum number of copies of an item if all the resources are used for this item. Another difference of this method is the batch addition of items. This method is shown to be effective in finding near optimal solutions.

Magazine and Oguz [39] proposed a dual heuristic. In this work, the algorithm starts with

all items in the knapsack. Each item is removed one at a time using the Lagrange multipliers. When feasibility is achieved, the algorithm starts adding the items back in the order of the objective value coefficient without violating the feasibility. Volgenant and Zoon [59] improved this algorithm by generating more than one Lagrange multiplier at a time and perturbing the Lagrange multipliers to improve the bound. Recently, Volgenant and Zwiers [60] improved this heuristic by using partial enumerations. In this study, the authors showed improvements in the quality of the solutions with some more computation time.

In Freville's review [19] on MKP problems, he mentioned that Oliva introduced constraint programming techniques into solving mixed integer programs and showed a decrease in time and number of nodes explored. We could not find this original work, however in recent years there have been many studies on this topic. Achterberg [1] used conflict analysis to improve the SCIP solver. Kilinc et al. explored dynamic search to reduce the size of the branching tree. However, these are studies on mixed-integer linear programming problems. Our work on canonical cuts resembles these works and focuses on MKP problems.

Commercial solvers such as CPLEX and SAS/OR are for any type of mixed integer linear programming problems. Commercial solvers are shown to be very effective on many problem types. These solvers have problem type specific components in addition to generic components. For example, in these solvers there are some cuts that can be used only for specific problem structures. Exploiting the problem structure and customizing the solving procedure according to problem type has been an effective trend.

Recently, limiting the search space has become popular. Using this idea, most promising results are reported by Vasquez and Hao [57], Vasquez and Vimont [58] and Raidl and Gottlieb [49]. Vasquez and Hao studied a hybrid approach based on tabu search. This method is improved recently by Vasquez and Vimont [58]. In this approach, the search space is reduced by using cardinality constraints. The LP solution is used as starting point for tabu search. In this study, the main improvement is gained by fixing the number of variables that can be assigned to their upper bound using cardinality constraints.

As part of reducing the search space approach, Fischetti and Lodi [16] proposed local branching and Danna et al. proposed induced neighborhood search [12]. Disjunctive separation of search space was proposed by Puchinger [46].

Balas and Zemel [7] proposed the core concept for MKP problem. Martello and Toth [41] and Pisinger [45] improved this concept that led to promising results. Puchinger et. al cooperated memetic and branch-and-cut algorithms for solving MKP in addition to core concept [47]. In a recent study, Puchinger et al. showed the results of using different efficiency factors with different core sizes [48]. They used the test library by Chu and Beasley. Promising results were reported for problems with five constraints. In the same study, Puchinger compared the performance of using CPLEX to memetic algorithms and using these two solving methods in

parallel while solving the core problems.

In conclusion, most promising results were reported with disjunctive search of reduced spaces. Fundamentally, our study uses a similar idea.

1.4 Dissertation Structure

In Chapter 2, we give theoretical background on canonical cuts for 0-1 problems. We reproduce some of the results from Balas and Jeroslow's seminal work [6]. In this chapter, we extend the theory to implement canonical cuts more effectively on MKP.

In Chapter 3, we propose a new cutting plane solver using the theory developed for canonical cuts. This solver is aimed at solving MKP problems. It has some components similar to commercial MILP solvers. The main difference of this solver is the implementation of canonical cuts.

Chapter 4 contains the experimental results. Results can be separated into two parts. The first part is the algorithmic experiments which focuses on the improvement of the cutting plane solver. The second part focuses on canonical cuts. Our implementations of canonical cuts are integrated into SAS/OR MILP solver. Experimental results are analyzed and canonical cuts are shown to improve the performance of the solver significantly over different test suites of MKP.

In Chapter 5, we study the multi-resource combinatorial auction problem. Due to the nature of this application, mathematical formulation is in the category of difficult MKP. In this chapter, we propose a test generation routine for this application and show the performance improvement of the SAS/OR MILP solver over the test problems of this application.

Chapter 6 states the conclusions and future work planned for this study.

Chapter 2

Canonical Cut Theory

Canonical hyperplanes and cuts were first introduced by Balas in 1970. However we could not find this work. The same author published these concepts in addition to some new results later in 1978 [6]. In literature, these cuts are also called *closure cuts* [32]. In constraint programming literature, they are called *clauses* [54]. Recently, these cuts have become popular for imposing the results of training the branching algorithms in solvers [1, 32]. In the next section, we describe the unit hypercube and some terminology for the vertices of the unit hypercube. Some fundamental theory on canonical hyperplanes and canonical cuts are reproduced in Sections 2.2 and 2.3 [6]. Theory on generalized hyperplanes is extended to derive stronger cuts and compare the strength of different canonical cuts. In the last section, we give information on the significance of these cuts to tighten the formulation of an arbitrary inequality.

2.1 Unit Hypercube and Its Properties

This section describes the unit hypercube and some terminology related to the hypercube [6]. The n-dimensional unit hypercube is defined as the following:

$$K = \{x \in R^n | 0 \leq x_j \leq 1, j \in N\}, \tag{2.1}$$

where $N = \{1, \ldots, n\}$.

Definition 2.1.1. *$x \in K$ is a vertex if n of the $2n$ inequalities defined in Formulation (2.1) are binding.*

The set of vertices is referred to as V. The following definitions are related to vertices of the hypercube.

Definition 2.1.2. *$x^i,\ x^j \in V$ are complementary if the intersection of the binding inequalities defining each of the two vertices is empty. The following relationship holds for complementary*

vertices

$$x^i = e - x^j$$

where e is an n-dimensional vector of which components are 1.

Definition 2.1.3. *$x^i,\ x^j \in V$ are adjacent if all but one of the binding inequalities defining each vertex are the same.*

Definition 2.1.4. *A line segment, l, that contains $x = \lambda x^i + (1-\lambda)x^j\ \forall\lambda \in [0,1]$ is called a closed edge or an edge.*

Definition 2.1.5. *A line segment, l, that contains $x = \lambda x^i + (1-\lambda)x^j$ for $\forall\lambda \in (0,1)$ is called an open edge.*

Definition 2.1.6. *A hyperplane which has $n-r$ of the $2n$ inequalities defined in (2.1) binding is called an r-dimensional face of the hypercube. It is referred to as F^r.*

The following information used in later parts of this chapter.

- A vertex is a *0-dimensional face*.
- An edge is a *1-dimensional face*.
- A face of order r is included in $n-r$ faces with dof $r+1$.
- A face of order r, is contained in $\binom{n-r}{s}$ faces of order $r+s$, for $1 \le s \le n-r$.

The following proposition becomes very useful when defining the separation problem in addition to the conclusion of some results. We represent the convex hull of a subset of vertices, R, with $C(R)$.

Proposition 2.1.7. *A point on an open edge, (x^1, x^2), is contained in $C(R)$ if and only if closed edge, $[x^1, x^2]$, is contained in $C(R)$.*

Proof. The closed edge contains the open edge by definition. If $C(R)$ contains the edge, then it also contains any point of the open edge. This concludes the proof for the *if* part of the proposition.

Assume that there exists $x \in (x^1, x^2) \cap C(R)$. As shown with the following equation, $x \in C(R)$ can be represented as a convex combination of the vertices included in $C(R)$.

$$x = \sum_{x^r \in R} \lambda_r x^r,\ \lambda_r \ge 0 \text{ and } \sum \lambda_r = 1 \tag{2.2}$$

Since x^1 and x^2 are on an edge, they are adjacent and differ only in one component. Let this component be the p^{th} component. Then $x_i = x_i^1 = x_i^2 = 1$ or 0 for all $i \ne p$. For all x^r for

which $x_j^r \neq x_i$ for $j \neq p$, $\lambda_r = 0$. Using this information, the above equation is reduced to the following form:

$$x = \lambda x^1 + (1 - \lambda)x^2 \;\; 0 < \lambda < 1 \tag{2.3}$$

$C(R)$ is the convex hull of some vertices. Assume that $\{1, 2\} \notin R$. Then x can not be in $C(R)$ since x can not represented without x^1 and x^2. □

2.2 Introduction to Canonical Hyperplanes

This section serves as introduction to a family of hyperplanes which are canonical [6]. The motivation of these hyperplanes is to separate the current region into two regions, one of which is the largest possible region that does not include any vertex of the hypercube except the ones that are used to derive the hyperplane. It is possible to find a hyperplane that contains only one of the vertices of the unit hypercube. The hyperplane that is tangent to the unit sphere at the given vertex has this property. For example, in 3-dimensional space, for vertex $v = (1, 1, 1)$, Equation (2.4) defines this hyperplane.

$$x_1 + x_2 + x_3 = 3 \tag{2.4}$$

The following hyperplane passes through all of the adjacent vertices of v.

$$x_1 + x_2 + x_3 = 2 \tag{2.5}$$

All adjacent vertices of the vertices on Hyperplane 2.5 pass through $x_1 + x_2 + x_3 = 1$, the only vertex left that is not on any of these hyperplanes is the origin. The origin is the only adjacent vertex to the vertices on $x_1 + x_2 + x_3 = 1$, and $x_1 + x_2 + x_3 = 0$ represents this vertex. As can be seen from this example, these hyperplanes can be used to discretize the space. As a side note, all of these hyperplanes belong to the same face.

Figure 2.1, shows Hyperplanes (2.4) and (2.5).

As can be seen in Figure 2.1, the half space on the side of (2.5) that does not include v provides the strongest separation of vertices of K from v. Generalization of this example gives rise to the development of the family of canonical hyperplanes. Following is the canonical cut that separates v from the rest of the vertices such that the side that includes v is the largest possible.

$$x_1 + x_2 + x_3 \leq 2 \tag{2.6}$$

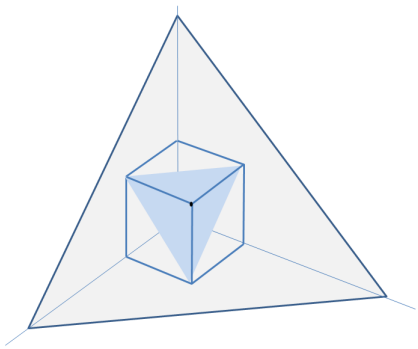


Figure 2.1: Canonical hyperplanes and canonical cuts

2.2.1 Edge Distance Function

To be able to define this family, Balas [6] proposes two new distance functions.

Definition 2.2.1. *Let $x^i = (x^i_1, x^i_2, \ldots, x^i_n)$ and $x^j = (x^j_1, x^j_2, \ldots, x^j_n)$ be two vertices of the unit hypercube. The edge distance, $d(x^i, x^j)$, between these two vertices is defined as the cardinality of the following index set:*

$$E = \{e : x^j_e \neq x^i_e\}$$

This distance function can be interpreted as the number of different binding constraints among the constraints stated in Equation (2.1). A chain between two vertices is defined as the collection of adjacent vertices that connects two vertices. It is possible to interpret the edge distance function as the length of a shortest chain between two vertices. It is also possible to define the same distance function between a face and vertex.

Definition 2.2.2. *The edge distance between a vertex, x, and a face, F, is defined as the minimum of the edge distances between all vertices on F and x as shown analytically in the following formulation:*

$$d(x, F) = min_{x^i \in F \cap V} d(x, x^i) \tag{2.7}$$

These distance functions are shown to satisfy the distance function requirements [6].

2.2.2 Canonical Hyperplanes

This subsection introduces the generalized form of the canonical hyperplanes in addition to some theoretical conclusions that show some nice properties of these hyperplanes [6].

Let F^r be an *r-dimensional* face of the unit hypercube K where $r \in \{1, \ldots, n-1\}$. All vertices on F^r have $n - r$ identical components. We use the following index sets for easy reference:

$$\begin{aligned} N(F^r)^+ &= \{i \in N | x_i = 1, \forall x \in F^r \cap V\} \\ N(F^r)^- &= \{i \in N | x_i = 0, \forall x \in F^r \cap V\} \\ N(F^r)^0 &= N - \left(N(F^r)^+ \cup N(F^r)^-\right) \end{aligned} \tag{2.8}$$

Equation (2.9) defines a canonical hyperplane of order r, $H(F^r)_d$, for $0 \leq d \leq n - r$, where d is the edge distance between vertices on $H(F^r)_d$ and F^r.

$$\sum_{i \in N(F^r)^+} x_i - \sum_{i \in N(F^r)^-} x_i = \left|N(F^r)^+\right| - d \tag{2.9}$$

For the example given in subsection 2.2, $v = (1, 1, 1)$ defines a face of order 0. $H(F^0)_1$ is $x_1 + x_2 + x_3 = 2$. $H(F^0)_2$ is $x_1 + x_2 + x_3 = 1$.

The order of a canonical hyperlane is defined as the order of the face that is used to define the hyperplane itself. Numerically the order of the hyperplane is same as the number of zero coefficients in Equation (2.9).

For future reference, the following propositions help us define the separation problem and complexity of the separation problem which is solved to validate the canonical cuts.

Proposition 2.2.3. *A vertex is on a canonical hyperplane, $H(F^r)_d$, if and only if the edge distance between the vertex and F^r is d.*

Proof. If $d(x, F^r) = d$, there are d components in x with either $x_i = 0$ for $i \in N(F^r)^+$ or $x_i = 1$ for $i \in N(F^r)^-$. All the points on F^r satisfy Equation (2.9) for $d = 0$. Since $n - d$ components of x are the same as all the points on F^r, x satisfies Equation (2.9).

To prove the *only if* part of the proposition, let x be a vertex that satisfy Equation (2.9). Vertices on F^r satisfies Equation (2.10):

$$\sum_{i \in N(F^r)^+} x_i - \sum_{i \in N(F^r)^-} x_i = \left|N(F^r)^+\right| \tag{2.10}$$

Since x_i can be either 1 or 0, the number of different components between a point that satisfies Equation (2.10) and another point that satisfies Equation (2.9) is d. So, $d(x, F^r) = d$. □

Proposition 2.2.4. *There are $2^r \binom{n-r}{d}$ vertices on a canonical hyperplane, $H(F^r)_d$.*

Proof. There are 2^r vertices on F^r. All vertices of F^r have $n-r$ identical components. If x is on $H(F^r)_d$, then $d(x, F^r) = d$. So d of the $n-r$ components that were identical for v on F^r are not identical for x on $H(F^r)_d$. $d(x, F^r) = d$ for x on $H(F^r)_d$. Using the proposition above, there are d different components between $H(F^r)_d$ and F^r. There are $\binom{n-r}{d}$ possible ways to choose these. So there are $2^r\binom{n-r}{d}$ vertices on $H(F^r)_d$. □

We continue this section by describing some of the nice properties of canonical hyperplanes. For the unit cube in 3-dimensional space, if there is a plane that passes through at least 3 vertices of the cube, then this plane is a canonical plane. This is not true for higher dimensions. In other words, in n-dimensional space, a hyperplane that passes through at least n vertices of the hypercube might not be a canonical hyperplane.

The following lemma and theorem will help us understand the strength of canonical hyperplanes and canonical cuts. With the following lemma and theorem we can say that any effort spent on proving the validity or invalidity of a canonical hyperplane or a cut does not provide any redundant information to solve the original problem.

Lemma 2.2.5. *A canonical hyperplane, H, contains a point of an open edge if and only if it contains the closed edge.*

Proof. If the whole edge is included on any plane, any point on the open edge is also included on the same plane, since the open edge is included in closed edge. This direction of the Lemma is true for any type of hyperplane.

Without losing generality, for any x on an open edge we can assume $0 < x_1 < 1$ and $x_j = 0$ or 1 for $j = 2 \dots n$. Since x is on H defined by Equation (2.9), the coefficient of x_1 must be 0. This implies that H is parallel to the edge that contains x which means it contains the whole edge. This concludes the proof of the *only if* part of the Lemma. □

Theorem 2.2.6. *The convex hull of vertices on a hyperplane, H, is represented by $C(V \cap H)$. Then following equivalence relationship holds for canonical hyperplanes.*

$$C(V \cap H) \equiv H \cap K \tag{2.11}$$

Proof. Assume that Equation (2.11) does not hold. Since K includes the convex hull of any combination of vertices, the right hand side can not be a proper subset of the left hand side. So we focus on proving the contradiction of the following equation.

$$H \cap K = C(V \cap H) \cup A, \tag{2.12}$$

where A is an arbitrary nonempty subset of K.

The equality above implies the existence of a vertex x on $H \cap K$ such that $n-1$ of the $2n$ inequalities of Equation (2.1) are tight and $x \notin V$. For x to be a vertex of $H \cap K$, it should lie on an open edge of K. This is a contradiction to Lemma 2.2.5. □

The properties described in Theorem 2.2.6 and Lemma 2.2.5 make canonical hyperplanes very appealing. These properties are not shared by arbitrary hyperplanes which contain n vertices of K. We adapted the following example from [6]. This example shows that a hyperplane might include a point on an open edge without containing the whole edge.

Example: The following is hyperplane E for the 4-dimensional hypercube.

$$x_1 + x_2 + x_3 + 2x_4 = 2$$

There are 4 vertices of the hypercube on this hyperplane. These are $(0,0,0,1)$, $(1,1,0,0)$, $(1,0,1,0)$ and $(0,1,1,0)$. This hyperplane contains $(1,0,0,\frac{1}{2})$ which is on an open edge between $(1,0,0,0)$ and $(1,0,0,1)$. However these vertices are not included on E. In addition, the convex hull of vertices on this hyperplane does not include $(1,0,0,\frac{1}{2})$.

2.3 Canonical Cuts

In this section, canonical cuts are explained and some results are reproduced [6]. Canonical cuts are used to refer to the halfspaces generated by canonical hyperplanes. Equation (3.4) defines the hyperspace defined by the canonical hyperplane, $H(F^r)_d$. $H(F^r)_d^+$ is used to represent this hyperspace for the rest of this section. The following inequality is referred to as the *canonical cut*:

$$\sum_{i \in N(F^r)^+} x_i - \sum_{i \in N(F^r)^-} x_i \leq \left|N(F^r)^+\right| - d \tag{2.13}$$

In this section, we state some conclusions on the relationships between the hyperspaces defined by the canonical hyperplanes.

So far, the adjacency concept is used to define adjacent vertices and adjacent faces. This concept can be extended to canonical hyperplanes.

Let $H(F_1^r)_{d_1}$ and $H(F_2^r)_{d_2}$ be two canonical hyperplanes. These are adjacent if $d_1 = d_2$ and one of the following conditions hold for an s.

- $N(F_2^r)^+ = N(F_1^r)^+ \cup \{s\}$ and $N(F_1^r)^- = N(F_2^r)^- \cup \{s\}$
- $N(F_1^r)^+ = N(F_2^r)^+ \cup \{s\}$ and $N(F_2^r)^- = N(F_1^r)^- \cup \{s\}$

In other words, all coefficients, except one, of the equations defining these two hyperplanes are the same. The one that is different is $+1$ in one and -1 in another. We keep the notation above for adjacent hyperplanes for the rest of this section.

Let the coefficient of x_s be $+1$ in the equality defining $H(F_1^r)_{d_1}$ and -1 in the equality defining $H(F_2^r)_{d_2}$. It is obvious that the *r-dimensional* faces, F_1^r and F_2^r, are parallel. $F_{1,2}^{r+1}$ has an order of $r+1$ and is defined by $n-r-1$ equations that define $H(F_1^r)_{d_1}$ and $H(F_2^r)_{d_2}$ in addition to $x_s = \alpha_s$ for $0 < \alpha_s < 1$.

Proposition 2.3.1. *If F_1^r and F_1^r are parallel faces both of which are contained in $F_{1,2}^{r+1}$ then $x \in V \cap H(F_1^r)_d^+ \cap H(F_2^r)_d^+ \Rightarrow x \in H(F_{1,2}^{r+1})_d^+$.*

Proof. If $x \in V \cap H(F_1^r)_d^+ \cap H(F_2^r)_d^+$ then the following equations hold for x:

$$\sum_{j \in N(F_1^r)^+} x_j - \sum_{j \in N(F_1^r)^-} x_j \leq \left|N(F_1^r)^+\right| - d \tag{2.14}$$

$$\sum_{j \in N(F_2^r)^+} x_j - \sum_{j \in N(F_2^r)^-} x_j \leq \left|N(F_2^r)^+\right| - d \tag{2.15}$$

The two hyperplanes defining Halfspaces (2.14) and (2.15) are adjacent, so without losing generality we assume $s \in N(F_1^r)^+ \cup N(F_2^r)^-$. Since $N(F_1^r)^+ = N(F_2^r)^+ \cup \{s\}$ and $N(F_2^r)^- = N(F_1^r)^- \cup \{s\}$, Inequalities (2.14) and (2.15) are rewritten as the following two inequalities.

$$\sum_{j \in N(F_2^r)^+} x_j + x_s - \sum_{j \in N(F_1^r)^-} x_j \leq \left|N(F_2^r)^+\right| + 1 - d \tag{2.16}$$

$$\sum_{j \in N(F_2^r)^+} x_j - x_s - \sum_{j \in N(F_1^r)^-} x_j \leq \left|N(F_2^r)^+\right| - d \tag{2.17}$$

Adding Inequalities (2.16) and (2.17) and dividing both sides by two, the following inequality is derived.

$$\sum_{j \in N(F_2^r)^+} x_j - \sum_{j \in N(F_1^r)^-} x_j \leq \left|N(F_2^r)^+\right| - d + \frac{1}{2} \tag{2.18}$$

x_j takes the values of 0 or 1. In addition to this, $N(F_2^r)^+ = N(F_{1,2}^{r+1})^+$ and $N(F_1^r)^- = N(F_{1,2}^{r+1})^-$. So Inequality (2.18) above can be reduced to:

$$\sum_{j \in N(F_{1,2}^{r+1})^+} x_j - \sum_{j \in N(F_{1,2}^{r+1})^-} x_j \leq \left|N(F_{1,2}^{r+1})^+\right| - d. \tag{2.19}$$

This concludes the proof. □

One interpretation of Proposition 2.3.1 is that the vertices included in a hyperspace defined by a lower dimensional face are also included in a higher dimensional face given that lower dimensional faces are parallel to each other and are contained in the higher dimensional face. It is possible to construct a similar type of proposition for the reverse diecrtion. In other words, given that some conditions hold, vertices contained in hyperspace defined by higher dimensional faces are included in a hyperspace defined by lower dimensional space. The following proposition describes this.

Proposition 2.3.2. *Let a face of order $r+1$ containing F^r be represented by F_s^{r+1} for $s = 1, \ldots, n-r$. Then the following statement holds:*

$$x \in V \cap \left\{ \cap_{s=1}^{n-r} H(F_s^{r+1})_d^+ \right\} \Rightarrow x \in H(F^r)_{d+1}^+. \tag{2.20}$$

Proof. The following inequalities are the general form of halfspaces defined by $H(F_s^{k+1})_d^+$:

$$\sum_{j \in N(F^r)^+ - \{s\}} x_j - \sum_{j \in N(F^r)^-} x_j \leq |N(F^r)^+| - 1 - d, \ s \in N(F^r)^+ \tag{2.21}$$

$$\sum_{j \in N(F^r)^+} x_j - \sum_{j \in N(F^k)^- - \{s\}} x_j \leq |N(F^r)^+| - d, \ s \in N(F^r)^- \tag{2.22}$$

There are $|N(F^r)^+| + |N(F^r)^-| = n - r$ halfspaces defined by Hyperplane (2.21) and (2.22). The following inequality is derived by adding all $n-r$ inequalities above:

$$(n-r-1) \left[\sum_{j \in N(F^r)^+} x_j - \sum_{j \in N(F^r)^-} x_j \right] \leq (n-r-1) \left[\left| N(F^r)^+ \right| - d \right] - d$$

Both sides are scaled down as follows:

$$\sum_{j \in N(F^r)^+} x_j - \sum_{j \in N(F^r)^-} x_j \leq \left| N(F^r)^+ \right| - d - \frac{d}{n-r-1}$$

Since $x_j = 0$ or 1 for $j \in N$, the right hand side can be replaced with its floor value.

$$\sum_{j \in N(F^k)^+} x_j - \sum_{j \in N(F^k)^-} x_j \leq \left| N(F^k)^+ \right| - d - 1$$

This concludes the proof. □

2.4 Extension of Theory

In this section we extend canonical cut theory to compare the strength of different canonical cuts. Our motivation is to implement canonical cuts as efficiently as possible. In this section we introduce the *depth-k* canonical cut. To the best of our knowledge, this is the first time this cut is studied.

Theorem 2.4.1. *Let F_1^r and F_2^s define two faces of orders r and s such that $r \neq s$. Then the following condition holds:*

$$H(F_1^r)_d^+ \subset H(F_2^s)_d^+,$$

if and only if

- $N(F_1^r)^+ \subset N(F_2^s)^+$
- $N(F_1^r)^- \subset N(F_2^s)^-$

Proof. Let the following relationships hold:

$$N(F_2^s)^+ = N(F_1^r)^+ \cup S^+$$
$$N(F_2^s)^- = N(F_1^r)^- \cup S^-.$$

The halfspace for F_1^r is given below:

$$\sum_{i \in N(F_1^r)^+} x_i - \sum_{i \in N(F_1^r)^-} x_i \leq \left|N(F_1^r)^+\right| - d$$
$$\sum_{i \in N(F_1^r)^+} x_i - \sum_{i \in N(F_1^r)^-} x_i \leq \left|N(F_2^s)^+\right| - \left|S^+\right| - d.$$

The halfspace for F_2^s is given below:

$$\sum_{i \in N(F_2^s)^+} x_i - \sum_{i \in N(F_2^s)^-} x_i \leq \left|N(F_2^s)^+\right| - d$$
$$\sum_{i \in N(F_1^r)^+} x_i + \sum_{i \in S^+} x_i - \sum_{i \in N(F_1^r)^-} x_i - \sum_{i \in S^-} x_i \leq \left|N(F_2^s)^+\right| - d.$$

The following inequality holds for all vertices of the unit hypercube:

$$\sum_{i \in S^+} x_i - \sum_{i \in S^-} x_i \leq |S^+|. \tag{2.23}$$

Due to Inequality (2.23), $H(F_1^r)_d^+$ is contained in $H(F_2^s)_d^+$. This concludes one direction of the proof.

For the other direction of the proof, we show the contradiction when $H(F_1^r)_d^+ \subset H(F_2^s)_d^+$ holds and the index sets satisfy the following conditions:

$$N(F_1^r)^+ = S^+ \cup P_1$$
$$N(F_2^s)^+ = S^+ \cup P_2$$
$$N(F_1^r)^- = S^- \cup R_1$$
$$N(F_2^s)^- = S^- \cup R_2$$

where $P_1 \cap P_2 = \emptyset$ and $R_1 \cap R_2 = \emptyset$.

The following two inequalities are the canonical cuts defined by F_1^r and F_2^s:

$$\sum_{i \in S^+} x_i - \sum_{i \in S^-} x_i + \sum_{i \in P_1} x_i - \sum_{i \in R_1} x_i \leq \left|S^+\right| + |P_1| - d \tag{2.24}$$
$$\sum_{i \in S^+} x_i - \sum_{i \in S^-} x_i + \sum_{i \in P_2} x_i - \sum_{i \in R_2} x_i \leq \left|S^+\right| + |P_2| - d \tag{2.25}$$

Let x be a solution that satisfies (2.25) such that $\sum_{i \in S^+} x_i - \sum_{i \in S^-} x_i = |S^+| + |P_2| - d + 1$, $x_i = 1$ for $\forall i \in P_1$ and $x_i = 0$ for $\forall i \in R_1$. This solution does not satisfy 2.24. This concludes the proof. □

As a side note, smaller half spaces imply stronger cuts. So we use Theorem 2.4.1 to generate cuts so that half spaces are the smallest without violating the restrictions we have. The following example illustrates this conclusion.

Example: The following index sets represent two faces of different orders in 5-dimensional space:

$$N(F_2^1)^+ = \{1,2\} \quad N(F_2^1)^- = \{3,4\} \quad N(F_2^1)^0 = \{5\}$$
$$N(F_1^3)^+ = \{1\} \quad N(F_1^3)^- = \{3\} \quad N(F_1^3)^0 = \{2,4,5\}.$$

The following two inequalities show the half spaces generated using the faces F_1^3 and F_2^1 for $d = 1$:

$$x_1 + x_2 - x_3 - x_4 \leq 1 \tag{2.26}$$
$$x_1 - x_3 \leq 0 \tag{2.27}$$

As seen above, the half space generated by F^2 is contained in the half space generated by F^1.

The analytical formulation for halfspace using canonical hyperplanes is given in Inequality (2.9). In literature, when $d = 1$, this formulation is referred to as canonical cuts. In this study we generalize the usage of canonical cuts to any d. From now on we call d the *depth* of the canonical cut.

The next two theorems provide us valuable information to generate strong cuts using parallel faces while changing the depth.

Theorem 2.4.2. *Let F^r define a face of order r with the index sets $N(F^r)^+$, $N(F^r)^-$ and $N(F^r)^0$. The following index sets describe parallel faces:*

- *F^r_k such that $N(F^r_k)^+ = N(F^r)^+ \cup \{k\}$, $N(F^r_k)^- = N(F^r)^- - \{k\}$, $N(F^r_k)^0 = N(F^r)^0$*
- *F^r_s such that $N(F^r_s)^+ = N(F^r)^+ - \{s\}$, $N(F^r_s)^- = N(F^r)^- \cup \{s\}$, $N(F^r_s)^0 = N(F^r)^0$.*

Let $H_{all}(F^r)^+_1 = H(F^r)^+_1 \cap \left(\bigcap_{t \in N(F^r)^+ \cup N(F^r)^-} H(F^r_t)^+_1 \right)$. Then the following relationship holds:

$$H(F^r)^+_2 \subset H_{all}(F^r)^+_1.$$

Proof. Let $x^0 \in H(F^r)^+_2$ then we show that $x^0 \in H_{all}(F^r)^+_1$.

The following inequality defines $H(F^r)^+_2$:

$$\sum_{i \in N(F^r)^+} x_i - \sum_{i \in N(F^r)^-} x_i \leq \left|N(F^r)^+\right| - 2. \tag{2.28}$$

Let $A = \sum_{i \in N(F^r)^+} x^0_i - \sum_{i \in N(F^r)^-} x^0_i$. An upper bound on A is $|N(F^r)^+| - 2$.

The following set of inequalities defines the depth-1 half spaces that form $H_{all}(F^r)^+_1$:

$$\sum_{i \in N(F^r)^+} x_i - \sum_{i \in N(F^r)^-} x_i \leq \left|N(F^r)^+\right| - 1 \tag{2.29}$$

$$\sum_{i \in N(F^r)^+ + \{k\}} x_i - \sum_{i \in N(F^r)^- - \{k\}} x_i \leq \left|N(F^r)^+\right| \tag{2.30}$$

$$\sum_{i \in N(F^r)^+ - \{r\}} x_i - \sum_{i \in N(F^r)^- + \{r\}} x_i \leq \left|N(F^r)^+\right| - 2. \tag{2.31}$$

If we substitute in the values for x^0, we get the following set of inequalities:

$$\begin{aligned} A &\leq \left|N(F^r)^+\right| - 1 \\ A + 2x^0_k &\leq \left|N(F^r)^+\right| \\ A - 2x^0_r &\leq \left|N(F^r)^+\right| - 2. \end{aligned}$$

Since $x^0_k \leq 1$ and $x^0_r \leq 1$, all inequalities above are satisfied.

The second part of the proof is proving the inclusion to be a proper subset. Let x^0 be such that $x^0_r = 1$, $x^0_k = 1$ and $\sum_{i \in N(F^r)^+} x^0_i - \sum_{i \in N(F^r)^-} x^0_i = |N(F^r)^+| - 1$. Then x^0 satisfies the inequalities (2.29), (2.30) and (2.31) but $x \notin H(F^r)^+_2$. So this concludes the proof for the proper subset relationship. □

Theorem 2.4.3. *Using the same notation as above, the following relationship holds:*

$$C(V \cap H(F^r)^+_2) \equiv C(V \cap H_{all}(F^r)^+_1).$$

Proof. In the proof of Theorem 2.4.2, we proved if $x \in H(F^r)^+_2$, then $x \in H_{all}(F^r)^+_1$ for x which is not necessarily a vertex of the unit hypercube.

So we only need to prove that if a vertex of K in $H(F^r)^+_2$, then it is contained in $H_{all}(F^r)^+_1$.

The second part of the proof is valid only for the vertices of K. Let x^0 be a vertex of K, $x^0 \in H_{all}(F^r)^+_1$ and $x^0 \notin H(F^r)^+_2$.

Then due to satisfying Inequality (2.29) and violating Inequality (2.28), the following equation holds:

$$\sum_{i \in N(F^r)^+ - \{r,k\}} x^0_i - \sum_{i \in N(F^r)^- - \{r,k\}} x^0_i + x^0_r - x^0_k = \left|N(F^r)^+\right| - 1. \tag{2.32}$$

Let $x^0_r - x^0_k = B$, then possible values for B, x^0_k and x^0_r are given below.

$$x^0_r = 1 \quad x^0_k = 1 \quad B = 0 \tag{2.33}$$
$$x^0_r = 0 \quad x^0_k = 0 \quad B = 0 \tag{2.34}$$
$$x^0_r = 1 \quad x^0_k = 0 \quad B = 1 \tag{2.35}$$
$$x^0_r = 0 \quad x^0_k = 1 \quad B = -1 \tag{2.36}$$

Given that the vertex x^0 is defined by Equation (2.32), if it satisfies Equations (2.33) and (2.36) then it violates Inequality (2.30); if it satisfies Equations (2.34) and (2.35), then it violates Inequality (2.31). So $x^0 \in H(F^r)^+_2$. This concludes the proof. □

Theorems 2.4.2 and 2.4.3 show the relationship between depth-1 and depth-2 canonical cuts. These results can be generalized for depth-k and depth-$(k+1)$ canonical cuts. The following two corollaries are generalizations of Theorems 2.4.2 and 2.4.3 to any depth.

Corollary 2.4.4. *Let* $H_{all}(F^r)^+_k = H(F^r)^+_k \cap \left(\bigcap_{t \in N(F^r)^+ \cup N(F^r)^-} H(F^r_t)^+_k \right)$, *where* $k \in \{0, \ldots, |N(F^r)^+|\}$. *Then the following relationship holds:*

$$H(F^r)^+_{k+1} \subset H_{all}(F^r)^+_k.$$

Corollary 2.4.5. *Using the same notation as in Corollary 2.4.4, the following relationship holds:*

$$C(V \cap H(F^r)^+_{k+1}) \equiv C(V \cap H_{all}(F^r)^+_k).$$

The proofs of these corollaries are similar to the proofs of Theorems 2.4.2 and 2.4.3.

2.5 Separation Problem

Let x^0 be a vertex of K. The following inequality defines the depth-1 canonical cut that cuts off this vertex:

$$\sum_{i \in N(F^0)^+} x_i^0 - \sum_{i \in N(F^0)^-} x_i^0 \leq \left|N(F^0)^+\right| - 1. \tag{2.37}$$

Given that x^0 is infeasible, the canonical cut given by Inequality (2.37) is valid.

The following inequality is the depth-2 canonical cut generated by the face containing x^0:

$$\sum_{i \in N(F^0)^+} x_i^0 - \sum_{i \in N(F^0)^-} x_i^0 \leq \left|N(F^0)^+\right| - 2.$$

The cut given by the Inequality (2.38) is valid given that there is no feasible solution on

$$\sum_{i \in N(F^0)^+} x_i^0 - \sum_{i \in N(F^0)^-} x_i^0 = \left|N(F^0)^+\right| - 1.$$

It is possible to implement canonical cuts at $x \in K - V$. One example of the implementation of depth-1 canonical cuts at fractional points is given in [55]. Soyster's implementation is applying the canonical cut at primal bound which is obtained from solving LP.

Let x^{LP} be the optimal solution of the LP relaxation. Let this point be on F^r_{LP}, then the following inequality defines the depth-1 canonical cut that cuts off F^r_{LP}:

$$\sum_{i \in N(F^r_{LP})^+} x_i^0 - \sum_{i \in N(F^r_{LP})^-} x_i^0 = \left|N(F^r_{LP})^+\right| - 1.$$

The separation problem is solving the feasibility problem on

$$\sum_{i \in N(F^r_{LP})^+} x_i^0 - \sum_{i \in N(F^r_{LP})^-} x_i^0 = \left|N(F^r_{LP})^+\right|. \tag{2.38}$$

The separation problem for the depth-2 canonical cut is, in addition to (2.38), the feasibility

problem on the following hyperplane:

$$\sum_{i \in N(F^r_{LP})^+} x^0_i - \sum_{i \in N(F^r_{LP})^-} x^0_i = \left|N(F^r_{LP})^+\right| - 1. \tag{2.39}$$

The following lemma generalizes the separation problem for depth-k canonical cut.

Lemma 2.5.1. *The depth-k canonical cut is valid at x given that no feasible solution exists on*

$$\sum_{i \in N(F^r_{LP})^+} x^0_i - \sum_{i \in N(F^r_{LP})^-} x^0_i = \left|N(F^r_{LP})^+\right| - p \tag{2.40}$$

for $p = 1, \ldots, k-1$.

We use implicit enumeration for solving the feasibility problem on the Hyperplane (2.40). Implicit enumeration is perceived to be an expensive procedure to solving integer programing problems due to the exponential nature of the complexity function. Since MKP has nonnegative integer objective and constraint coefficients, and implicit enumeration is a simple method to program, we prefer implicit enumeration as a solver for the feasibility problem. It is possible to use any MILP solver to solve the feasibility problem on canonical hyperplanes. In this study, although implicit enumeration is efficient for this problem type, we still impose a limit on the number on function evaluations to make the usage of canonical cuts more practical. Some experimental results will be shown in later sections on how to limit the computational effort while generating the canonical cuts. The following corollary describes the relationship between complexities of separation problems for depth-1 and depth-2 canonical cuts that are parallel to the same face.

Theorem 2.5.2. *The complexity of solving the separation problem of* $H_{all}(F^r)^+_1$ *is equivalent to solving the separation problem of* $H(F^r)^+_2$.

Proof. Let $H(F^r)^-_d$ represent the following hyperspace:

$$\sum_{i \in N(F^r)^+} x^0_i - \sum_{i \in N(F^r)^-} x^0_i \geq \left|N(F^r)^+\right| - 1. \tag{2.41}$$

It is possible to represent the separation problem as proving the infeasibility of the Hyperspace (2.41). The following interpretation of the conclusion from Theorem 2.4.3 shows the equivalence relationship between the complexities of the separation problems of $H_{all}(F^r)^+_1$ and $H(F^r)^+_2$:

$$C(V \cap H(F^r)^-_2) \equiv C(V \cap H_{all}(F^r)^-_1).$$

This concludes the proof. □

The following example shows the conclusion of Theorem 2.5.2 in five dimensional space.

Example: Let x^0 be a point in five dimensional hypercube. Followings are the index sets for x^0:

$$\begin{aligned} N(F^2)^+ &= \{1,2\} \\ N(F^2)^- &= \{3\} \\ N(F^2)^0 &= \{4,5\}. \end{aligned}$$

The separation problem for $H_{all}(F^2)_1^+$ is solving the feasibility problem on the following hyperplanes:

$$x_1 + x_2 - x_3 = 2 \tag{2.42}$$
$$-x_1 + x_2 - x_3 = 1 \tag{2.43}$$
$$x_1 - x_2 - x_3 = 1 \tag{2.44}$$
$$x_1 + x_2 + x_3 = 3. \tag{2.45}$$

The complexity of this problem is $O(4 \times 2^2)$.

The separation problem for $H(F^r)_2^+$ is the feasibility problem on the following two canonical hyperplanes:

$$x_1 + x_2 - x_3 = 2 \tag{2.46}$$
$$x_1 + x_2 - x_3 = 1. \tag{2.47}$$

The feasibility problem on the Hyperplane (2.46) is included as part of showing the validity of $H_{all}(F^2)_1^+$. The feasibility on the Hyperplane (2.47) is equivalent to showing the feasibility of Hyperplanes (2.43), (2.44) and (2.45).

Therefore, solving the separation problem for $H_{all}(F^2)_1^+$ and $H(F^r)_2^+$ are equivalent, while $H(F^r)_2^+$ provides a stronger cut than $H_{all}(F^2)_1^+$.

2.6 Relation Between Canonical Cuts and Arbitrary Inequalities

We finish this section by stating the relationship between arbitrary inequalities and canonical cuts. Imposing an arbitrary inequality on a hypercube is equivalent to imposing a collection of set covering inequalities which can be shown to be canonical cuts [6]. Interested readers can investigate the proof of this conclusion in [6]. The reason we include this conclusion is to show our motivation to study these cuts. If we add the correct canonical cuts then it is possible to

convert the original mathematical formulation to a formulation with canonical cuts. Since if a point on an open edge is contained on a canonical hyperplane then the whole edge is included, it might be possible to converge to the tightest formulation possible by adding the right canonical cuts without getting any redundant information while solving the separation problem. The following list summarizes our motivation to study canonical cuts.

- Any constraint can be represented with a set of canonical cuts.
- Canonical cuts pass through the vertices of the unit hypercube.
- Given some information on the infeasibility of certain vertices, canonical cuts provide the strongest possible reduction in the feasible region.

Chapter 3

Cutting Plane Solver for the Multi-dimensional Knapsack Problem (MKP)

The initial motivation for this study was to develop a specialized cutting plane solver to solve MKP which would address the shortcomings of the branching approach that is widely used in literature. For the cutting plane solver, we base our approach on geometric insights and use experimental results to evaluate our approach. This section describes the thought process of developing the specialized cutting plane solver for MKP.

3.1 Components of the MKP Solver

In this section we describe the components of the cutting plane solver we are proposing. The main idea of this solver, as it is in all of the integer programming solvers, is decreasing the gap between the objective value of the best known feasible integer solution and the bound on the solution. Reducing this gap to zero provides both necessary and sufficient conditions on the termination.

3.1.1 Initial Primal and Dual Bounds

The first step is to find an initial integer solution. We use a greedy approach. For the MKP problem, assigning all x_is to 0 gives a feasible solution. This solution is the origin of the hyperspace in which the feasible region is defined. Starting from the origin, among x_is at their lower bounds, the x_i with the largest impact on the objective value is moved from its lower bound to its upper bound while keeping the feasibility of the solution. This step is repeated as

long as there is an x_i which has these properties. Figure 3.1 illustrates this algorithm.

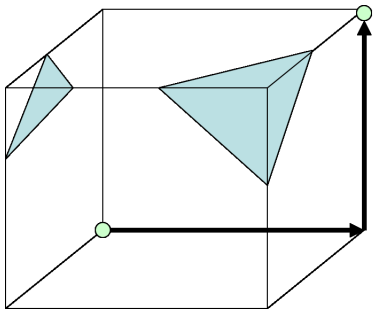


Figure 3.1: Greedy approach to find initial solution

The objective value of the solution found by the greedy algorithm provides a primal bound. We use the primal bound to impose an objective cut as stated by the following inequality.

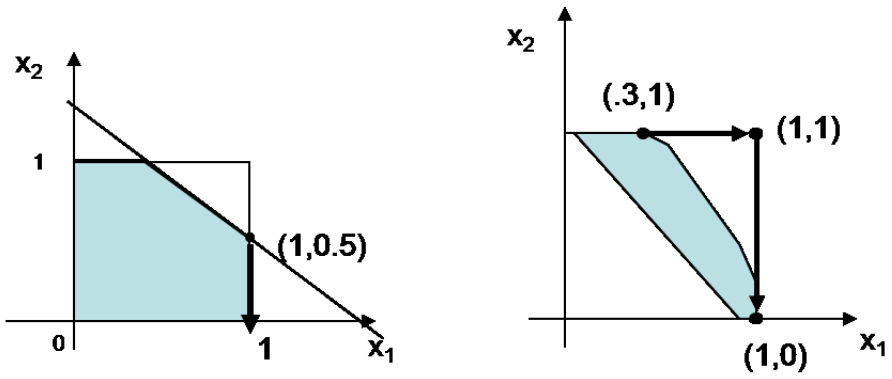
$$c^T x \geq \text{Primal Bound} + 1 \tag{3.1}$$

To get a dual bound on the problem, the integrality requirement is relaxed. The relaxed problem is an LP, and the objective value of the LP solution is a widely used dual bound for any integer linear programming problem. As a side note, if the solution that provides the dual bound is an integer solution, then it is an optimal solution for the original problem as this implies the gap is 0. We use the upper bounded simplex algorithm to solve the LP relaxation. More detail on this algorithm can be found in [13].

3.1.2 Primal Bound During the Solve

In this subsection, we describe how we continue finding better feasible integer solutions. We propose a primal heuristic approach. At the dual bound, all the constraints except the integrality constraints are satisfied. The number of x_is in the basis gives the number of integrality constraints that are violated. Each violated integrality constraint is tried to be restored using upper bounded simplex pivot. We use two types of pivots. During a type-1 pivot, each fractional x_i is pivoted to either its upper bound or lower bound. Type-1 pivot requires LP feasibility to be maintained all the time. Figure 2(a) illustrates this pivoting rule.

As shown in Figure 2(b), type-1 pivot might not be feasible for some cases. A type-2 pivot suggests doing two consecutive pivots. During the first pivot, the feasibility requirement for the original linear constraints is relaxed while the integrality constraint for one of the fractional x_i is restored. In the second pivot, all violated original linear constraints are restored by pivoting



(a) Type 1 pivot

(b) Type 2 pivot

Figure 3.2: Primal heuristics

an integral x_j from the current bound to the opposite bound. During the first pivot, the number of fractional x_is decreases by at least one; during the second pivot, the number of fractional x_is stays the same or in some cases decreases.

3.1.3 One Binding Constraint

In this subsection, we focus on some special cases to help us understand how to approach MKP using canonical hyperplanes. The first case we investigate is when at the current solution, only one of the original linear constraints is binding. At such a point, we look for a new primal bound by using type-1 pivots. Figure 3.3 shows an example of such a case. The current solution is shown with the black dot. Green dots show the visited integer points with the type-1 pivot. If any of the green dots provides a feasible integer solution then we have a new primal bound. Green dots are the only vertices which are on the same canonical hyperplane, $x_1 + x_3 = 2$, as the current solution as shown in Figure 3(b). If none of these vertices is feasible then the following canonical inequality provides the strongest valid cut using the given information:

$$x_1 + x_3 \leq 1. \tag{3.2}$$

Inequality (3.2) is the depth-1 canonical cut that cuts off the current solution. For the current solution, vertices visited via type-2 pivots are shown as red dots in Figure 3.4. The canonical hyperplane that passes through these vertices is shown in Figure 4(b). This hyperplane is the same as the canonical hyperplane that defines the depth-1 cut. All the other vertices of the cube are in the following hyperspace, which is the depth-2 canonical cut generated from the

face the current solution is on:

$$x_1 + x_3 \leq 0. \tag{3.3}$$

With one binding constraint, looking for the feasible type-1 pivots is equivalent to solving the separation problem for the depth-1 canonical cuts. In addition to the type-1 pivots, looking for the feasible type-2 pivots is equivalent to solving the separation problem for the depth-2 canonical cuts. Corollaries 3.1.1 and 3.1.2 generalize these conclusions.

Corollary 3.1.1. *Let x be a feasible LP solution on a face, F^1, of order 1. Starting from x, if none of the type-1 pivots visits a feasible integer solution then $H(F^1)_1^+$ states a valid cut.*

Proof. Let the current solution be x^0 on F^1. We use $N(F^1)^+$, $N(F^1)^-$ and $N(F^1)^0$ to represent the index sets of the variables at 1, 0, and a fractional value correspondingly. The following represents the index sets of the solutions that might be visited by the type-1 pivot.

- $N(F^0)^+ = N(F^1)^+ \cup N(F^1)^0$, $N(F^0)^- = N(F^1)^-$ and $N(F^0)^0 = \emptyset$.
- $N(F^0)^+ = N(F^1)^+$, $N(F^0)^- = N(F^1)^- \cup N(F^1)^0$ and $N(F^0)^0 = \emptyset$.

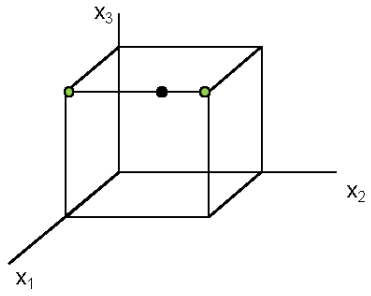
The solutions shown with the index sets above represent all the possible F^0 contained in F^1. Proposition 2.3.1 implies that we have visited all the vertices on F^1. $H(F^1)_1^+$ is the canonical hyperspace that represents the remainder of the vertices. This concludes the proof. □

Corollary 3.1.2. *Let x be a feasible LP solution on a face, F^1, of order 1. Starting from x, if none of the type-1 and type-2 pivots visit a feasible integer solution then $H(F^1)_2^+$ states a valid cut.*

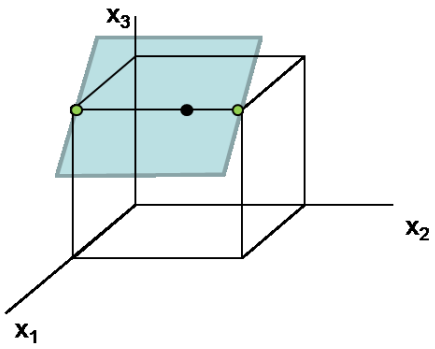
Proof. Type-1 pivots visit all the vertices on F^0s contained in F^1, as explained in the proof for Corollary 3.1.1. Type-2 pivots visit all the vertices on the faces that have an edge distance of 1 with the vertices visited during the type-1 pivot. The index sets for these faces are given below.

- $N(F^0)^+ = N(F^1)^+ \cup N(F^1)^0 - \{j\}$ and $N(F^0)^- = N(F^1)^- \cup \{j\}$, for $j \in N(F^1)^+$.
- $N(F^0)^+ = N(F^1)^+ \cup N(F^1)^0 \cup \{i\}$ and $N(F^0)^- = N(F^1)^- - \{i\}$, for $i \in N(F^1)^-$.
- $N(F^0)^+ = N(F^1)^+ - \{j\}$ and $N(F^0)^- = N(F^1)^- \cup N(F^1)^0 \cup \{j\}$, for $j \in N(F^1)^+$.
- $N(F^0)^+ = N(F^1)^+ \cup \{i\}$ and $N(F^0)^- = N(F^1)^- \cup N(F^1)^0 - \{i\}$, for $i \in N(F^1)^-$.

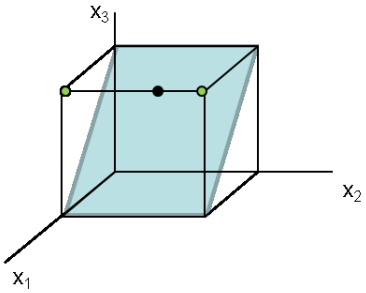
All the vertices on F^1 and all the vertices on faces with one edge distance from the vertices on F^1 are visited. The remainder of the vertices are in the region defined by $H(F^1)_2^+$. This concludes the proof. □



(a) Vertices visited by type-1 pivot

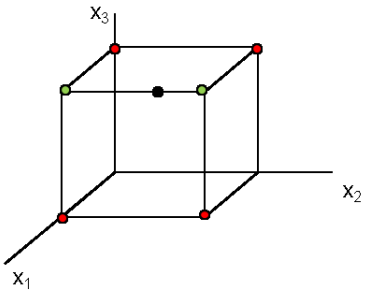


(b) Canonical hyperplane that passes through the current solution

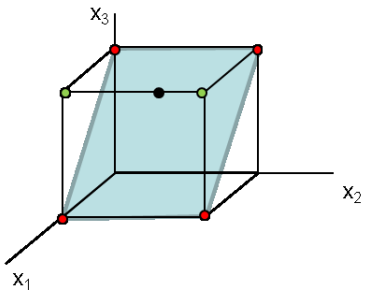


(c) Canonical hyperplane that provides the strongest cut

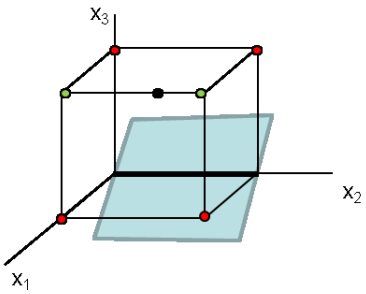
Figure 3.3: One binding constraint



(a) Vertices visited by type-1 and type-2 pivots



(b) Canonical hyperplane that passes through the vertices visited by type-2 pivots



(c) Canonical hyperplane that provides the strongest cut

Figure 3.4: One binding constraint

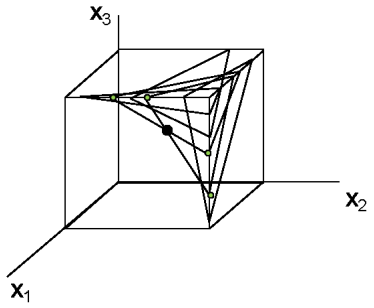
3.1.4 Two or More Binding Constraints

In this subsection, we extend the arguments in Subsection 3.1.3 for the solutions with two binding constraints.

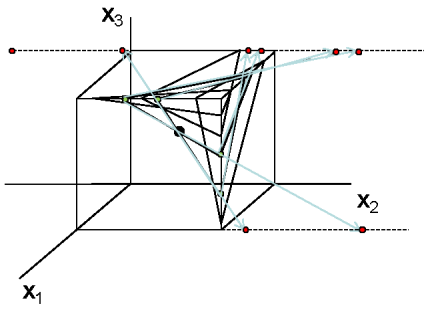
Corollary 3.1.3. *Let x be a feasible LP solution on a face, F^2, of order 2. Given that starting from x, none of the type-1 and type-2 pivots visits a feasible solution, it is not necessarily true that F^2 is infeasible.*

Proof. Assume that none of the solutions visited via type-1 and type-2 pivots are feasible and this implies the infeasibility of the face contains x, F^2. We show the contradiction of this argument with an example. Let x be the solution shown with a black dot in Figure 3.5. There are two binding constraints at this solution. Green dots are the solutions visited via type-1 pivots as shown in Figure 5(a). Red dots are the possible solutions visited by type-2 pivots as shown in Figure 5(b). The depth-1 canonical cut that cuts off the face x is on is shown as the blue hyperplane in Figure 5(c). However, this cut is not valid since there are feasible vertices on F^2. This concludes the proof. □

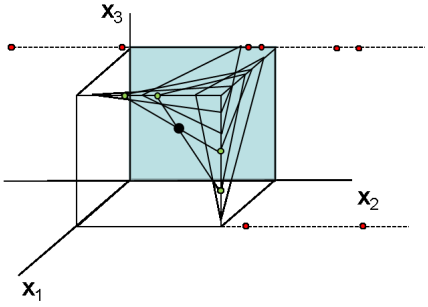
This conclusion is also valid for cases with more than two binding constraints. For the solutions that are on faces of order greater than 1, the infeasibility information of type-1 and type-2 pivots is not enough to conclude the infeasibility of the separation problem.



(a) Solutions visited by type-1 pivots



(b) Solutions visited by type-2 pivots



(c) Closest canonical hyperplane that cuts off the face that contains the current solution

Figure 3.5: Two binding constraints

3.1.5 Implementation of Canonical Cuts

The following inequality describes depth-k canonical cuts that cut off the face, F^r that x^0 is on:

$$\sum_{i \in N(F^r)^+} x_i - \sum_{i \in N(F^r)^-} x_i \leq \left|N(F^r)^+\right| - k. \tag{3.4}$$

We use the index sets $N(F^r)^+$, $N(F^r)^-$ and $N(F^r)^0$ that are described previously.

As part of the cutting plane solver, we implement depth-1 canonical cuts first. PB refers to the objective value of the current incumbent solution. Before adding the depth-1 canonical cut onto the problem, we solve the following feasibility problem:

$$\begin{aligned} Ax &\leq b && (3.5)\\ cx &\geq \text{PB} + 1 && (3.6)\\ \sum_{i \in N(F^r)^+} x_i - \sum_{i \in N(F^r)^-} x_i &= \left|N(F^r)^+\right|. && (3.7) \end{aligned}$$

As already mentioned, the MKP is an NP-hard problem. The mathematical formulation given by (3.5), (3.6), and (3.7) is also an NP-hard problem. However, the complexity of solving this problem is less than the complexity of solving the original problem given that $\|N(F^r)^0\| < n$. The complexity of this feasibility problem is $O(2^{\|N(F^r)^0\|})$, while the complexity of the original problem is $O(2^n)$. Equation (3.7) implies the following:

- $x_i = 1, \forall i \in N(F^r)^+$
- $x_i = 0, \forall i \in N(F^r)^-$.

Let $\overline{x}$ represent a vertex in the original space such that:

$$\begin{aligned} x_i &= 1, \forall i \in N(F^r)^+ \\ x_i &= 0, \forall i \in N(F^r)^- \cup \quad N(F^r)^0 \end{aligned}$$

We define a reduced space by fixing the x_i for $i \in N(F^r)^+$ to 1 and x_i for $i \in N(F^r)^-$ to 0. Let x' be a point in the reduced space. The feasibility problem is reduced to the following mathematical formulation:

$$\begin{aligned} Ax' &\leq b - A\overline{x} \\ cx' &\geq \text{LB} + 1 - c\overline{x}. \end{aligned} \tag{3.8}$$

We use implicit enumeration to solve the reduced feasibility problem. In implicit enumeration, all possible solutions are systematically evaluated without explicitly evaluating all of the feasible

region [33]. The implicit enumeration we implement is equivalent to branch and bound on the reduced space. We follow a tree structure during implicit enumeration. After fixing some values on the tree and assigning 0 to the unfixed variables, if the solution is not feasible then the rest of the tree can be pruned. When we have a feasible solution, we compare the objective value with the objective value of the incumbent solution. If it is better than the current incumbent solution, we update the incumbent solution and continue searching the rest of the tree. We use *depth-first* search to reduce the memory requirement.

Although implicit enumeration performs much better than explicit enumeration, it might still take very long to converge. One problem with implicit enumeration, and in all integer programming solvers, is the exponential increase in time to converge. We implement the implicit enumeration in MATLAB first to see the relationship between time it takes to converge and the complexity of the problem. Figure 3.6 shows this relationship. We use the results shown in

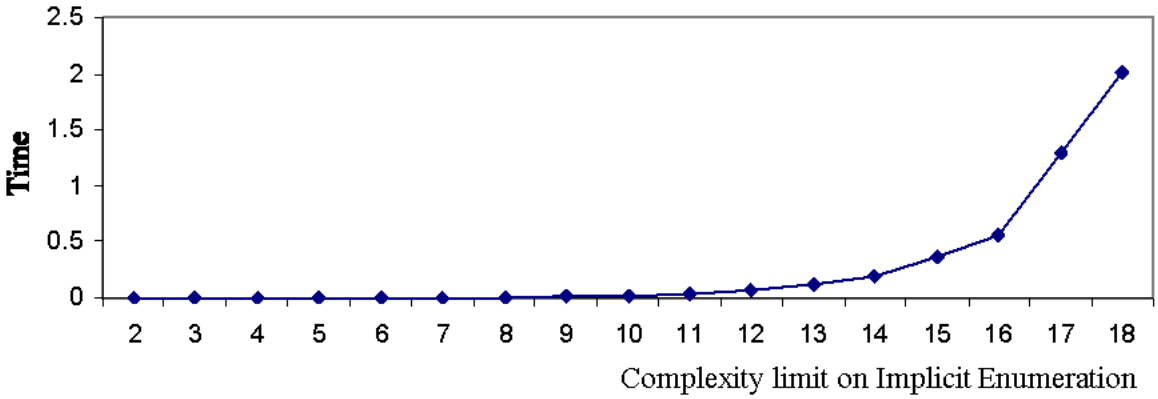


Figure 3.6: Implicit enumeration, time vs. complexity

Figure 3.6 to decide the complexity limit we want to solve in the implementation of the depth-1 canonical cuts. Integration of the cut into the cutting plane solver is shown in Figure 3.7. The flow shown in Figure 3.7 is a generic flow we use for adding any type of canonical cut. We impose a preferred limit (p) on complexity of the separation problem. Using the type-1 and type-2 pivots, the solver tries to find a feasible solution such that the complexity limit on the separation problem for the corresponding canonical cut is p. If there is no feasible pivot that leads to such a solution, the solver still solves the separation problem.

The same logic is used for integrating the depth-2 and depth-3 canonical cuts into the cutting plane solver. The following two mathematical formulations define the separation problem for

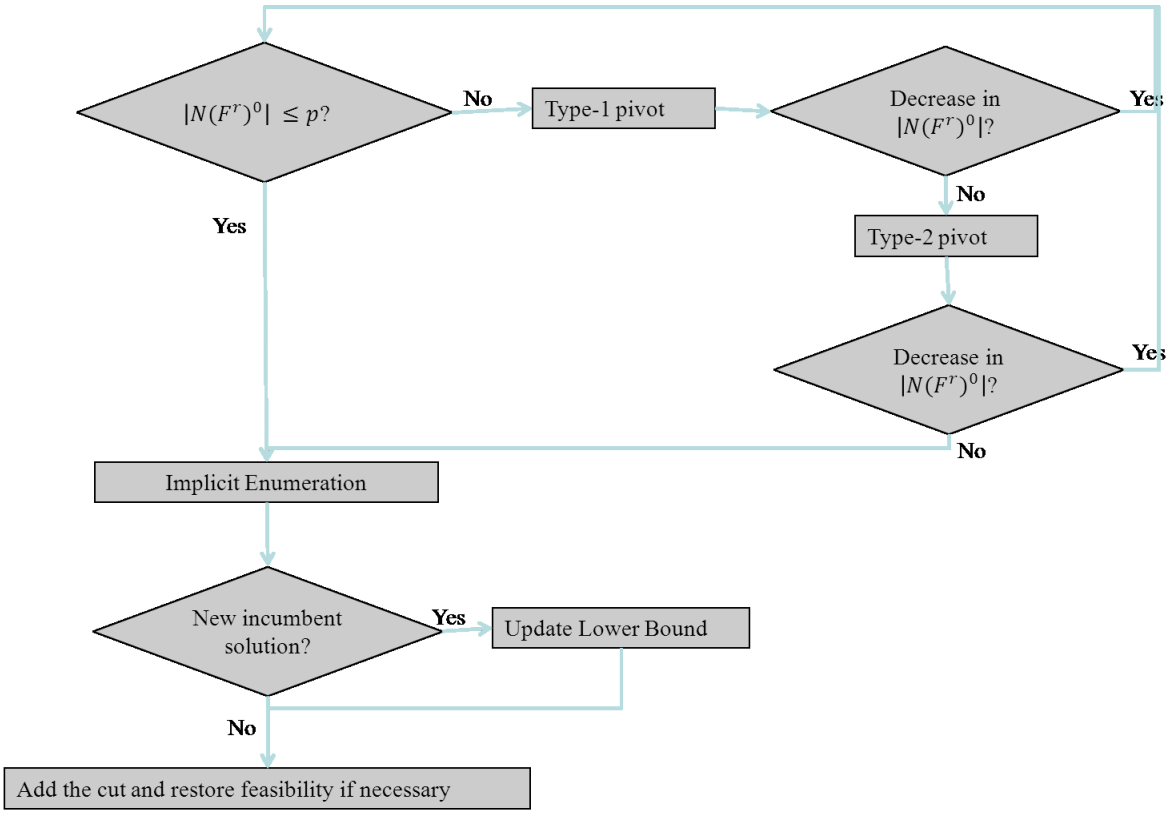


Figure 3.7: Canonical cut integration into the cutting plane algorithm

depth-2 canonical cut:

$$\begin{aligned} Ax &\leq b \\ cx &\geq \text{LB} + 1 \\ \sum_{i \in N(F^r)^+} x_i - \sum_{i \in N(F^r)^-} x_i &= \left|N(F^r)^+\right| \end{aligned} \tag{3.9}$$

and

$$\begin{aligned} Ax &\leq b \\ cx &\geq \text{LB} + 1 \\ \sum_{i \in N(F^r)^+} x_i - \sum_{i \in N(F^r)^-} x_i &= \left|N(F^r)^+\right| - 1. \end{aligned} \tag{3.10}$$

Formulation (3.9) is the same as the separation problem of the depth-1 canonical cut. We solve

the reduced problem described by Formulation (3.8). For solving the mathematical program (3.10), we solve the feasibility problems which have the same complexity as Formulation (3.8). The number of such feasibility problems is $|N(F^r)^+| + |N(F^r)^-|$. The following list defines $\overline{x}$ for each of the feasibility problems.

- $x_i = 1$, for $i \in N(F^r)^+ \cup \{s\}$ where $s \in N(F^r)^-$ and $x_j = 0$, for $j \in N(F^r)^- - \{s\}$
- $x_i = 1$, for $i \in N(F^r)^+ - \{s\}$ and $x_j = 0$, for $j \in N(F^r)^- \cup \{s\}$ where $s \in N(F^r)^+$

The separation problem for depth-3 canonical cuts is the union of the separation problem for the depth-2 canonical cut that is defined on the same face, F^r, and the feasibility problems of the reduced spaces that are defined by the following $\overline{x}$s:

- $\overline{x_i} = 1$, for $i \in N(F^r)^+ \cup \{s, k\}$ where $s, k \in N(F^r)^-$ and $\overline{x_j} = 0$, for $j \in N(F^r)^- - \{s, k\}$
- $\overline{x_i} = 1$, for $i \in N(F^r)^+ - \{s, k\}$ and $\overline{x_j} = 0$, for $j \in N(F^r)^- \cup \{s, k\}$ where $s, k \in N(F^r)^-$
- $\overline{x_i} = 1$, for $i \in N(F^r)^+ \cup \{s\} - \{k\}$ where $s \in N(F^r)^-$ and $k \in N(F^r)^+$ and $\overline{x_j} = 0$, for $j \in N(F^r)^- - \{s, k\}$.

3.1.6 Redundancy Detection

In this subsection, we propose the redundancy detection rule for canonical cuts. This feature is to avoid carrying the cuts that never become active as a result of being dominated by the other cuts. The complexity of this procedure is $O(n)$. Let the following two inequalities represent two cuts:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 \tag{3.11}$$

$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2. \tag{3.12}$$

We define the following index sets which describe the relationships between the coefficients of Inequalities (3.11) and (3.12):

$$P = \{i : a_{2i} > a_{1i}\}$$
$$R = \{i : a_{1i} > a_{2i}\}$$

Cut (3.12) dominates Cut (3.11) if either one of the following conditions holds:

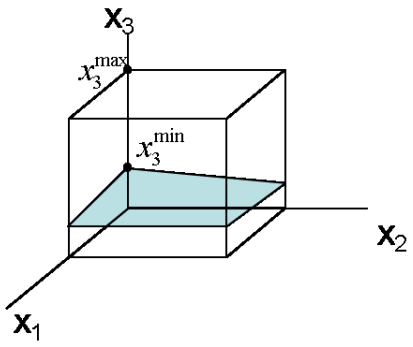
- $b_2 - b_1 \geq sum_{i \in P}(a_{2i} - a_{1i})$, if $b_2 \geq b_1$
- $|b_2 - b_1| \geq sum_{i \in P}(a_{1i} - a_{2i})$, if $b_1 \geq b_2$.

These rules can be used for any type of cuts given that the original problem is a 0-1 IP.

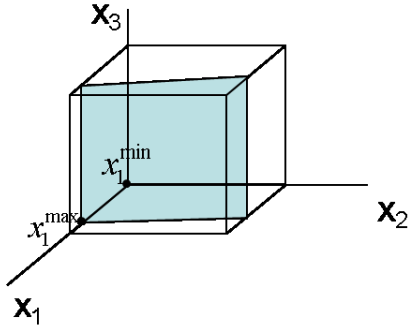
We present the statistics that show the discrepancy between total number of cuts generated and total number of non-redundant cuts in the computational experiment section to show the benefit of this feature.

3.1.7 Basic Prepocessor

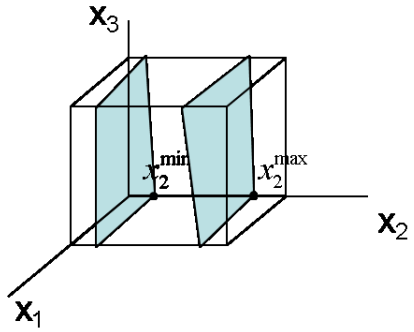
This section describes a preprocessing routine to speed the convergence. As described before, a variable can take values of 0 or 1. Three conditions to fix a variable are shown in Figure 3.8. If the minimum value of a variable is larger than 0 then it can be fixed at its upper bound. An example for this case is shown in Figure 8(a). If the maximum value of a variable is less than 1 then it can be fixed at 0 as shown in Figure 8(b). If the minimum value of a variable is larger than 0 and the maximum value of it is smaller than 1, then this implies that there is no feasible solution left in the remaining region, so this provides the sufficient condition to terminate. In every iteration, after adding the cut, for each variable, we call the LP solver two times to determine whether a variable is fixed or not. The fixing logic is given in Figure 3.1.7.



(a) x_i, fixed at upper bound



(b) x_i, fixed at lower bound



(c) No feasible solution

Figure 3.8: Fixing variables

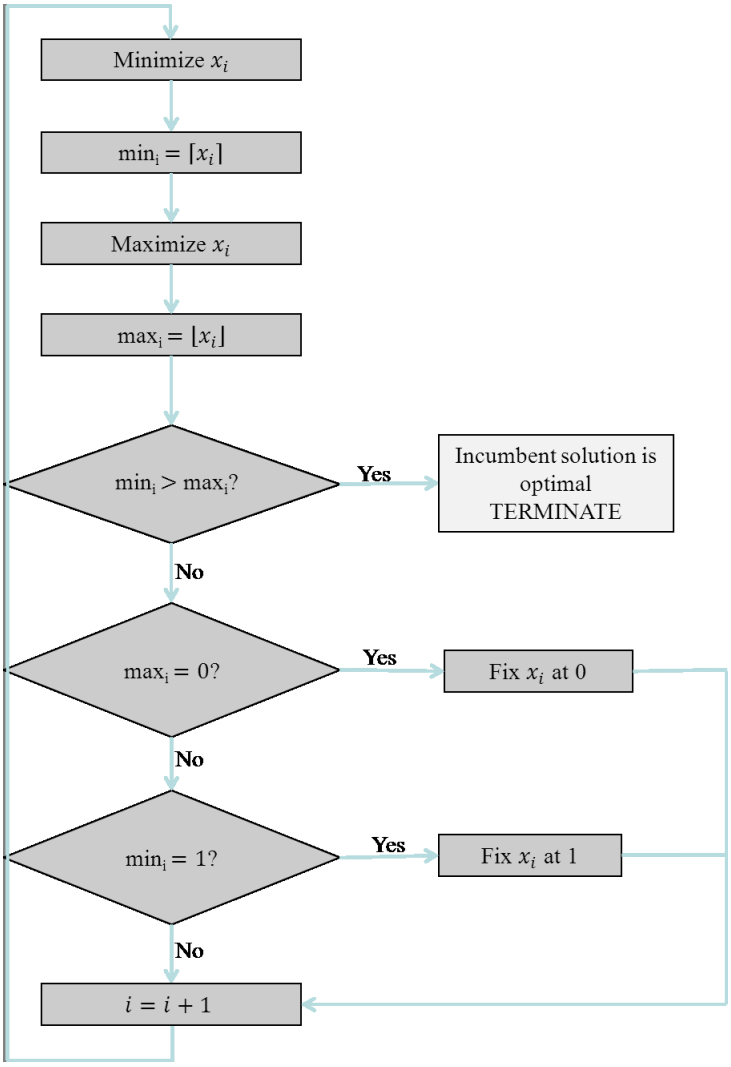


Figure 3.9: Fixing routine

3.2 Pure Cutting Plane Algorithm

Components described in the previous sections are used to develop a pure cutting plane algorithm. When we started this study, since MKP has only positive coefficients, our belief was that a pure cutting plane algorithm with an efficient cut which respected the problem structure would be more effective. We propose the pure cutting plane algorithm shown in Figure 3.10. The cutting plane solver starts by finding an incumbent solution using the greedy approach described in Subsection 3.1.1. Then an objective cut to improve the initial bound is added.

In the next step, we use the upper bounded simplex method to solve the LP relaxation in order to get a dual bound on the problem. If the LP solver cannot find a feasible solution then this implies the optimality of the initial incumbent solution and this is sufficient to conclude the convergence. The next condition to decide the convergence is whether the upper bound from the LP relaxation is less than or equal to the objective value of the incumbent solution. If so, then the optimality of the incumbent solution is proved. If not, the search for new integer solutions with better objective value than the objective value of the incumbent solution using the pivot rules starts. There are two usages of type-1 and type-2 pivots. The first one is searching for a new incumbent solution. The second one is finding feasible LP solutions on the faces of lower order if the complexity limit for the separation problem is violated on the current face. If the type-1 and type-2 pivots visit a feasible integer solution, the incumbent solution and the objective cut are updated accordingly.

The separation problem is defined at the solution on a face of which order is closest to the complexity limit. Implicit enumeration is used for solving the separation problem. If a new incumbent solution is found during implicit enumeration, the incumbent solution and the objective cut is updated with the new primal bound. We apply the canonical cut to the problem without evaluating its quality. With this cut, we are guaranteed to cut at least the LP solution which is on the face that is used to define the canonical cut. After adding the cut, if the current solution violates the new cut, then feasibility is restored.

The next step is to check whether any variable can be fixed or the integrality of a variable is causing infeasibility. Having all variables fixed at either their upper bound or lower bound provides a new incumbent solution and is a sufficient condition for convergence. Having a variable that is fixed at both its upper bound and lower bound implies infeasibility of the remaining region and provides a sufficient condition to terminate. This flow is repeated until any of the termination conditions are met.

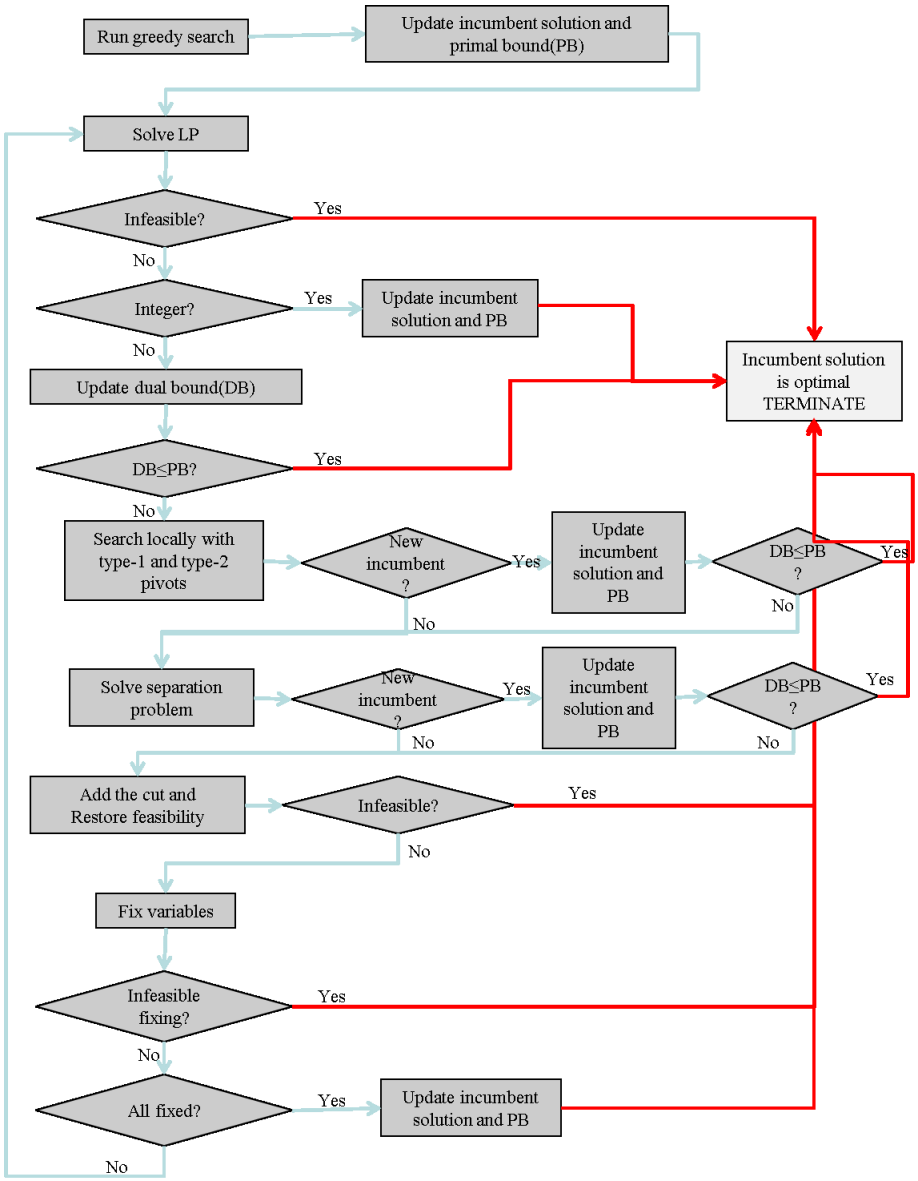


Figure 3.10: Pure cutting plane algorithm with canonical cuts

3.3 Cutting Plane Algorithm with n-Way Branching on Cardinality Constraints

To increase the efficiency of the solver, we implement space discretization to cut some fractional region. We use cardinality constraints for this purpose. The cardinality constraint is a special case of canonical hyperplanes. The following equality describes the generic form of this constraint:

$$\sum_{i \in P} x_i = r$$

where $P \subseteq \{1, \ldots, n\}$ and $r \in \{1, \ldots, n\}$.

It is possible that there exists a feasible integer solution on each possible cardinality hyperplane such as $\sum_{i \in P} x_i = n$ and $\sum_{i \in P} x_i = 1$. Due to the constraints and the objective cut on the problem, this is very unlikely. We propose to get an upper bound and lower bound on the feasible values of r to increase the efficiency of the solver. First we maximize $\sum_{i \in N} x_i$ for $N = \{1, \ldots, n\}$ over the feasible LP region. Let the maximum value of this summation be $k_{\max}$. Then we minimize it, and let the value of this summation be $k_{\min}$. Due to the integrality requirement on x_is, the following relationship is implied.

$$\lceil k_{\min} \rceil \leq \sum_{i \in N} x_i \leq \lfloor k_{\max} \rfloor$$

Then we use the cutting plane solver with the following constraint for each integer k' between $\lceil k_{min} \rceil$ and $\lfloor k_{max} \rfloor$.

$$\sum_{i \in N} x_i = k'$$

This approach resembles a branching scheme. We have not encountered this type of branching in the literature. We call this n-way branching.

As shown in [4], having multiple rounds of cut generation before starting the branching increases the performance. This is referred to as root node solving in solvers. We propose the same setting for our solver. This introduces a new parameter which determines when to start branching. Figure 3.11 shows the flow of the solver when the branching is on.

After the root node solving, the LP solver is called twice to get the upper ($\lfloor k_{max} \rfloor$) and the lower ($\lceil k_{min} \rceil$) bounds for the right hand side of the cardinality constraints. Then the cutting plane solver searches the feasible region on the cardinality hyperplane with right hand side of $\lceil k_{min} \rceil$. The primal bound and the incumbent solution during local solving is the same as the global primal bound and global incumbent. Each of the cardinality hyperplanes with integral right hand side less than or equal to $\lfloor k_{max} \rfloor$ is locally solved by the cutting plane solver.

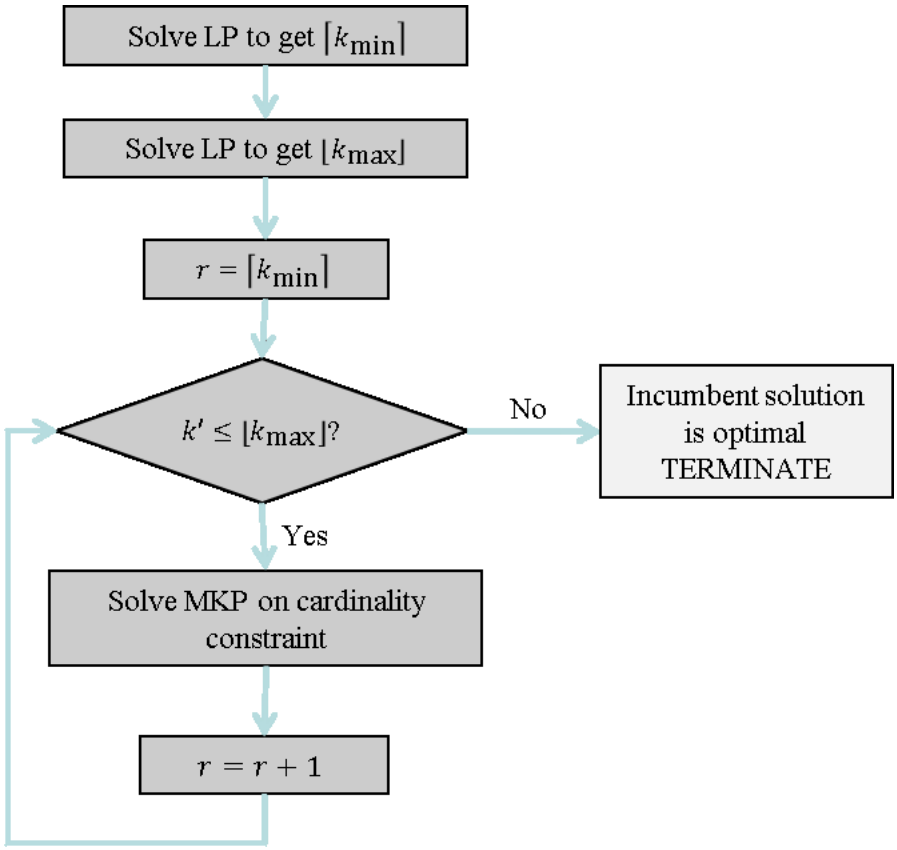


Figure 3.11: Cutting plane algorithm with n-way branching

3.4 Finite Convergence and Validation

In this section, we give the proofs of finite convergence and validity.

Theorem 3.4.1. *The proposed pure cutting plane algorithm with the depth-1 canonical cut converges in finite number of steps.*

Proof. There are finite number of faces of the unit hypercube in n-dimensional space. There are $2^{n-r}\binom{n}{n-r}$ faces of order r. At each iteration, the separation problem is solved for one distinct face and the upper bound on the complexity is to go through all faces and generate a canonical cut for each. The following equation gives the total number of faces of any order:

$$\sum_{r=0}^{n} \binom{n}{n-r} 2^{n-r} \quad = \quad \sum_{k=0}^{n} \binom{n}{k} 2^k = (1+2)^n = 3^n.$$

For a face of order 0, the separation problem is already solved. So the upper bound on the number iterations is $3^n - 2^n$. This concludes the proof. □

Corollary 3.4.2. *The proposed pure cutting plane algorithm with depth-k canonical cut converges in finite number of steps.*

The proof of this corollary is similar to the proof of Theorem 3.4.1.

As a side note, this upper bound for solving the problem given in the proof of Theorem 3.4.1 is a very loose upper bound for the MKP. Having the same complexity as the upper bound (going through all faces) implies the feasibility of the vertex at which all x_is are at their upper bounds and this creates a trivial instance. For this instance the optimal solution of the LP relaxation is an integer solution.

Corollary 3.4.3. *The proposed cutting plane algorithm with n-way branching and with depth-1 canonical cut converges in finite number of steps.*

Proof. For n-dimensional space, the upper bound on the number of the feasible cardinality hyperplanes to be searched is n. The number of distinct faces on the cardinality hyperplane with a right hand side $k \leq n$ is given below:

$$\sum_{r=0}^{k} \binom{n}{n-r} 2^{n-r} \leq 3^n.$$

So the upper bound on the number of distinct faces is $n3^n$. This concludes the proof. □

We continue with proving the validity of the proposed algorithms.

Theorem 3.4.4. *Convergence of the pure cutting plane solver implies the incumbent solution to be optimal to the original problem.*

Proof. Let x^* be the optimal integer solution and z^* be the corresponding objective value. There are three conditions that provide the sufficient condition for convergence. The first one is:

- Dual Bound (DB) − Primal Bound (PB) < 1.

Assume that $z^* \neq$ PB. Then this implies $z^* \geq$ PB $+ 1$ and $z^* >$ DB. This is a contradiction. The second one is:

- Infeasible fixing of x_s.

Assume $z^* >$ PB and x_s is either 0 or 1 at the optimal solution. This is a contradiction. The third one is:

- Feasibility cannot be restored after cut is applied.

Let the depth-k canonical cut that creates the infeasibility be on a face of order r. The following inequality is the depth-k canonical cut:

$$\sum_{j \in N(F^r)^+} x_j - \sum_{j \in N(F^r)^-} x_j \leq \left|N(F_1^r)^+\right| - k. \tag{3.13}$$

Since Inequality (3.13) is a disjunctive inequality, x^* is in either $\sum_{j \in N(F^r)^+} x_j - \sum_{j \in N(F^r)^-} x_j \leq |N(F_1^r)^+| - k$ or $\sum_{j \in N(F^r)^+} x_j - \sum_{j \in N(F^r)^-} x_j > |N(F_1^r)^+| - k$. The former is infeasible and the infeasibility of the latter is proved as part of the separation problem. This is a contradiction. This concludes the proof. □

Corollary 3.4.5. *Convergence of the the pure cutting plane solver with n-way branching implies the incumbent solution to be optimal to the original problem.*

The proof of Corollary 3.4.5 follows the same logic as the proof of Theorem 3.4.4.

3.5 Motivation for a More Sophisticated Solver

In this study, we started with a pure cutting plane solver. To make the usage of canonical cuts more efficient and increase the speed of convergence, we added some preprocessing techniques. Fixing and redundancy detection are the features that address this need. We implement a branching scheme using a special case of canonical cuts called cardinality cuts. All of these additions make the solver more efficient and faster. This is the same logic commercial solvers

such as CPLEX and SAS/OR MILP use. Commercial solvers have many other components that make them very efficient. This makes the performance comparison between commercial solvers and our solver not comparable. One thing different in our solver is various depth level canonical cuts. So we continue this work by focusing more on canonical cuts. We integrate various depth level canonical cuts within SAS/OR MILP solver which is known to be a very strong commercial solver. In the next section, we give brief information on SAS/OR MILP solver.

3.6 SAS/OR MILP Solver

Information in this section is from SAS/OR 9.3 Mathematical Programming User's Guide [51]. As part of the SAS/OR product, the OPTMODEL procedure includes a modeling language to model mathematical programming problems, and solvers for different types of problems such as linear and nonlinear programs. The mixed-integer linear programming (MILP) solver in the OPTMODEL procedure is used to solve the standard mixed-integer linear program.

The MILP solver has an LP-based branch and bound algorithm. The solver has presolving techniques, cut generation and primal heuristics to increase the efficiency of the general algorithm. Since our implementation of the canonical cuts might improve both primal and dual bounds, we give some more information on primal heuristics and cuts of the SAS/OR MILP solver. Cutting plane strategies are used to improve the dual bound while primal heuristics are used to improve the primal bounds. The main goal of primal heuristics is to find good primal bounds early in the search tree so that a large portion of the tree is pruned early. Techniques such as rounding, iterative rounding, and neighborhood searches are some of the primal heuristics used in the solver. As part of the solving, it is possible to turn off the heuristics or apply them at different levels such as default and aggressive. In our experiments, we use the solver with default heuristic setting.

Cuts are used to tighten the LP formulation of the problem so that the discrepancy between LP and MILP gets smaller. There are two types of cuts generated during the solving which are the generic cuts and the structured cuts. The generic cuts are:

- Gomory mixed integer
- Lift-and-project
- Mixed integer rounding
- Mixed lifted 0-1
- Zero-half.

The second category is the structured cut category. The solver looks for a certain structure to implement these cuts. If only part of the problem has the desired structure, cuts are generated for this part. Structured cuts are:

- Cliques
- Flow cover
- Flow path
- Generalized upper bound cover
- Implied bound
- Knapsack cover.

It is important to note that the solver takes advantage of the problem structure. It is possible to change the cut adding strategy from none to aggressive. For our testing purposes, we use the default cut adding strategy which is tuned to work well for most instances. Cuts are expected to improve the bounds and decrease the relative gap, in other words speed up the convergence.

Each cut has rules to be turned on and off. A cut type can be turned off permanently or temporarily. During each cut generation round, all of the cuts are visited. Depending on the active cut generating rules, one or more cuts from each cut type are generated. The solver decides about adding a certain number of cuts based on the quality of the cuts. It is possible that a cut is activated or deactivated during the solving. This decision is usually based on the performance of the cut. Some computational results for cut management of MOPS solver are shown in a study by Wesselmann [18].

Details on integration of canonical cuts to SAS/OR MILP solver are given in the computational experiments chapter.

Chapter 4

Computational Experiments

In this chapter we present the computational results. We start this chapter by describing the test suites. The rest of this chapter is divided into two parts. The first part focuses on the experiments of the different features of the proposed cutting plane solver that is described in Chapter 3. These experiments help us to determine the direction of our study. Significant performance improvements are gained from the canonical cuts. Using this conclusion, for the second part of the experiments we focus on the canonical cuts. We integrate the proposed canonical cuts into the SAS/OR MILP solver to measure the performance changes with the various implementations of the canonical cuts.

4.1 Description of Test Suites

For the first part of the computational experiments, we use randomly generated test problems. The coefficients of these problems are not correlated. In the literature, many preliminary studies used randomly generated uncorrelated test problems [19]. Especially early studies for the exact algorithms report the performance measures for these type of test problems. We use the uncorrelated randomly generated test problems for the probing experiments using the proposed exact algorithm. However, to the best of our knowledge, there is no widely used test suite of uncorrelated test problems in the literature. Due to this, we generate the test problems. For the second part of the computational results we use the test generator proposed by Freville and Platue [21] to generate a wide range of test problems. We also report the results of the proposed implementations over ORLIB problems [11] for benchmarking purposes. In the next two sections we give more details on these test suites.

4.1.1 Uncorrelated Test Problems

There are three types of coefficients that are generated. A constraint coefficient is represented with a_{ij}, an element of right hand side vector is represented with b_i, and an objective coefficient is represented with c_j. For a problem of size $n \times m$, there are n objective coefficients, m right hand side coefficients and $m \times n$ constraint coefficients. All of these coefficients are nonnegative integers. We prefer to keep these problems easy so that we can do probing experiments quickly. Another motivation to generate easy problems for the initial stage of the research was our unawareness of the limitations of our algorithm. To make the problems easy, we randomly generate uncorrelated coefficients within a small range. The following formulations satisfy our requirements:

$$\begin{aligned} a_{ij} &= U[0,10] + 1 \\ c_j &= U[0,10] + 5 \\ b_i &= U[0,50] + 51 \end{aligned}$$

Where $U[0,10]$ represents a number that is randomly generated between 0 and 10. As can be seen from the formulations, the coefficients are uncorrelated and the feasible solution set is small due to the small range of coefficients. Since our focus is on dense problems, we avoid zero coefficients. For that purpose, we add a constant term to each coefficient. We keep the size of the feasible region small by imposing a right hand side that is independent of the dimension of the space.

Table 4.1: Parameters for uncorrelated test suite

n	m
30	15
50	25
70	35
100	50

This test generation requires two parameters which are the number of variables (n), and the number of constraints (m). Table 4.1 states all the parameter combinations we use. For most of the probing experiments, we use $n = 30$ and $m = 15$ due to the limitations of our solver. We use the other combinations which require longer test runs to measure the potential of our approach to be used as a heuristic.

We mostly use point estimates such as mean to measure the performance changes with this

test suite. For the point estimates to be representative, we generate 100 problems for each combination. We justify using point estimates with the nature of the probing experiments. The purpose of the probing experiments is to determine the focus of our research, so doing an extensive analysis for measuring the performance changes is not feasible within our timeline.

4.1.2 Correlated Test Problems

For MKP, correlation between the objective function and constraint coefficients increases the hardness of the problem [20, 21]. Freville and Plateau [21] propose a procedure to generate test cases such that the objective function is correlated with the constraints. In this test generation procedure the difficulty of the test problems can be adjusted by changing some parameters. Given that the test problem is generated for n variables and m constraints, the following formulation describes this test generator:

$$\begin{aligned} a_{ij} &= U[0, 1000] \\ c_j &= \sum_{i \in M} \frac{a_{ij}}{m} + KU[0, 1] \\ b_i &= \alpha \sum_{j \in N} a_{ij}. \end{aligned} \tag{4.1}$$

Parameters K and α control the difficulty of the problem. Parameter K is called the correlation factor. As K decreases, correlation increases and problems become harder.

As mentioned before, we use uncorrelated test problems for probing experiments. However when we observe a dramatic improvement with various depth level canonical cuts, we use correlated test problems to see whether same level improvement is achieved over more difficult test problems as part of the probing experiments. Due to the limitations of our solver, we experiment only with $n = 30$, $m = 15$, $\alpha = 0.15$ and $K = 500$.

In addition to the probing experiments, we use this test generator to generate a variety of test problems and use these problems to evaluate the performance changes with proposed implementations within the SAS/OR MILP solver which are explained in Subsection 4.3.1. Our purpose is to find the most effective implementation and parameter for certain type of problem. For these test problems $n \in \{50, 100, 150\}$, $m \in \{15, 25, 50\}$, $\alpha \in \{0.15, 0.25, 0.50\}$ and $K = 500$. There are 27 combinations and we generate 20 problems per combination. These parameters give us different shaped constraint matrices with feasible regions of different size.

Chu and Beasley [11] use this test generator to generate problems with the parameters such that $K = 500$, $\alpha \in \{0.25, 0.50, 0.75\}$, $m \in \{5, 10, 30\}$ and $n \in \{100, 250, 500\}$. They generate 10 problems of each combination. The total number of problems is 270. These problems are part of the OR Library. Since then, these test problems became the standard way to evaluate

approaches for solving the MKP. We also use these problems for benchmarking purposes.

4.2 Experimental Results of Cutting Plane Solver

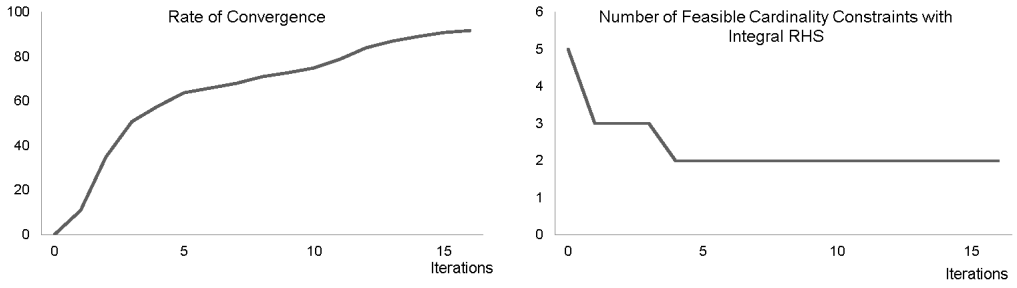
In this part of the dissertation, we present the initial experiments that focus on the proposed cutting plane solver. In these experiments, we evaluate the performance changes when new features are added or when the setting of a feature changes. We investigate the performance measure distributions and use the results to propose two heuristics. The last subsection of this part focuses more on the canonical cuts and serves as an introduction to the experiments of the canonical cuts within the SAS/OR MILP solver.

For the experiments of this section, we mostly use the uncorrelated test problems. For the last experiment which measures the performance improvements of various depth level canonical cuts we also use the correlated test problems. For the development and tuning of the algorithm, we only use test cases of 30 variables and 15 constraints. We use Windows 32 bit machine with Intel core duo processor, CPU of 2.50GHz and 4GB RAM. As a side note, for the experiments of this section, the depth-1 canonical cut is the default cut unless otherwise is mentioned.

4.2.1 Root Node Solve

In this section, we analyze the experimental results of the pure cutting plane solver over the uncorrelated test suite and use these results to determine when to finish the root node solving and start the n-way branching. There are two concerns that determine how we analyze the results. The first one is that for some problems, branching might not be necessary. Since branching might require more computation than the pure cutting plane, we prefer branching on more difficult problems. The second one is providing better bounds and a smaller feasible region for branching to improve the convergence. Figure 1(a) shows the rate of convergence versus number of iterations. More than fifty percent of the problems is solved within five iterations. After five iterations, convergence increase rate drops. Figure 1(b) shows the number of feasible cardinality constraints with integral right hand side versus number of iterations. This plot provides an indication for remaining effort for n-way branching. After five iterations, there is no improvement for the remaining number of constraints.

Results of this analysis depend on the problem size. We limit our experiments over 30 variables and 15 constraints and conclude that n-way branching starts after five iterations. However for more robust implementations, extensive testing over a variety of test problems is recommended.



(a) Convergence improvement

(b) Change in the number of feasible cardinality hyperplanes with integral right hand side

Figure 4.1: Determination of the end of root node solving

4.2.2 Several Features of the Pure Cutting Plane Solver

The purpose of the experiments in this section is to evaluate the performance of the different settings of the cutting plane solver. We use the uncorrelated test suite with $n = 30$ and $m = 15$. We compare the different features and the parameters of the algorithm. One feature is whether to use the pure cutting plane solver or the cutting plane solver with n-way branching. We also analyze the change in performance with various limits on the complexity of the separation problem of depth-1 canonical cuts. One expensive feature is the fixing. So we experiment on whether doing the fixing in an aggressive or moderate level. Moderate fixing is running the fixing procedure consecutively for each variable once. Aggressive fixing is running fixing consecutively however when a variable is fixed, restarting the fixing from the first variable. Figure 4.2 shows the average cpu time comparison for different solver versions for changing the complexity limits. There is a significant improvement when n-way branching is added on to the cutting plane solver. There is a slight change between the aggressive and the moderate fixing. For the pure cutting plane solver, having aggressive fixing performs better while for the cutting plane solver with n-way branching moderate fixing works better most of the time. Increasing the upper bound on complexity of the separation problem decreases the time to converge significantly. However, time decrease stops after a certain limit. For the pure cutting plane solver, the best performing limit is 16 while it is 15 with the n-way branching. We believe the slight decrease in the best performing complexity limit with n-way branching is due to the faster convergence of the algorithm with the n-way branching. In other words, n-way branching makes the algorithm more efficient so the benefit of higher complexity limit is not compensated.

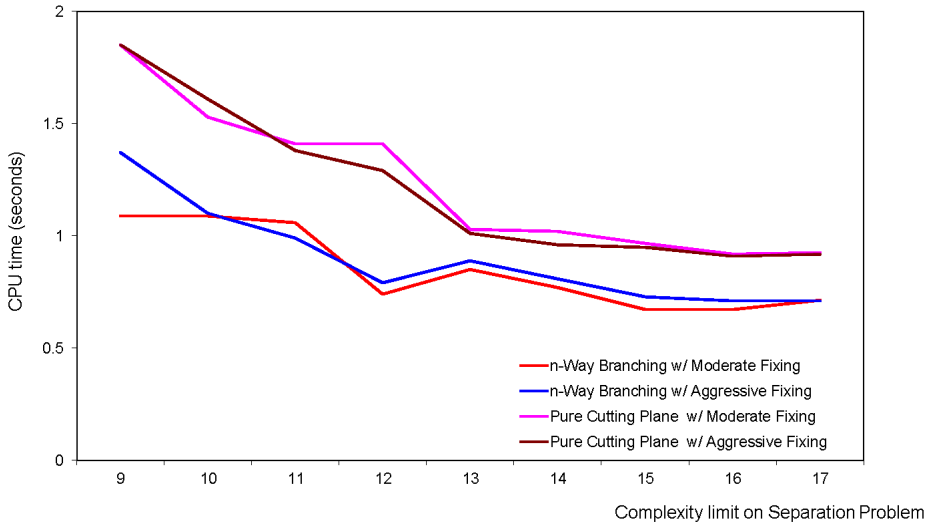


Figure 4.2: Pure cutting plane versus n-way branching

4.2.3 Face Variations For the Canonical Cut Generation

In the previous experiments we conclude that the changes in the complexity limit of the separation problem solving improved the time to convergence significantly. In this section we focus on another aspect of the canonical cut generation. We analyze the change in the effectiveness of the canonical cuts when there are variations for the face it is generated from. We impose this variation by picking different orders of type-1 and type-2 pivots. This feature is coded as an option on the solver. The default setting is the consecutive order. For consecutive order, variables are pivoted in the order of their indices. The following list states the possible values for this option:

- Consecutive
- Random
- Maximum objective value decrease
- Minimum objective value decrease.

When the option is set to random, we generate a random number that determines the next variable to be pivoted. For the maximum and the minimum objective value decrease, at each pivoting step, we evaluate the objective value change for all possible pivots and pick the one that has the maximum or minimum among all the values depending on the option

Table 4.2: Performance measures with different pivot orders

	Cuts Added	Non-redundant Cuts	CPU Time
Pivot option	Avg.	Avg.	Avg.
Consecutive	5.68	3.82	0.65
Random	5.69	3.83	0.68
Max Decrease	5.69	3.84	0.68
Min Decrease	5.70	3.84	0.68

value. We use uncorrelated test problems with $n = 30$ and $m = 15$ for this experiment. In this experiment, the base settings are the best performing ones from Subsection 4.2.2. The n-Way branching is turned on with the separation problem of complexity limit of 15. Table 4.2 gives the numerical results of this experiment. As changes in this feature might affect the cut generation, we analyze the average number of cuts generated and the average number of the nonredundant cuts in addition to the changes in CPU time. We use mean as the point estimate of the performance. There is no significant change for these performance measures. The best performing setting is the default setting. We believe this is due to the extra computation required for other settings. Especially last two options require more time than other two options. Although, there is some expectation for random order to perform better, since the test problems are randomly generated, there is no significant change in the performance. There is a slight increase in the average number of non-redundant cuts with the options other than consecutive setting. Since consecutive order is always the same from iteration to iteration, it is more likely to have redundant cuts with the consecutive setting. This result coincides with this intuition. Overall we conclude that face variation does not improve the performance.

4.2.4 Performance Measure Distributions

In the previous sections, we analyze the average values of the performance measures to evaluate the effectiveness of the different features of the cutting plane solver. In this subsection, we analyze the performance measure distributions of the solver with the best parameter settings concluded by the previous experiments. We use the uncorrelated test suite with 30 variables and 15 constraints. We use the following performance measures to analyze the results of the experiments.

- CPU time (Figure 4.3)
- The number of nonredundant cuts at termination (Figure A.1)
- The number of cut generations (Figures 2(a) and 2(b))

- The number of solution evaluations (Figures 3(a) and 3(b))
- The number of pivots (Figures 4(a) and 4(b))

As part of this subsection we include only the plot of the distribution for the CPU time to converge. We include the rest of the plots in Appendix A.

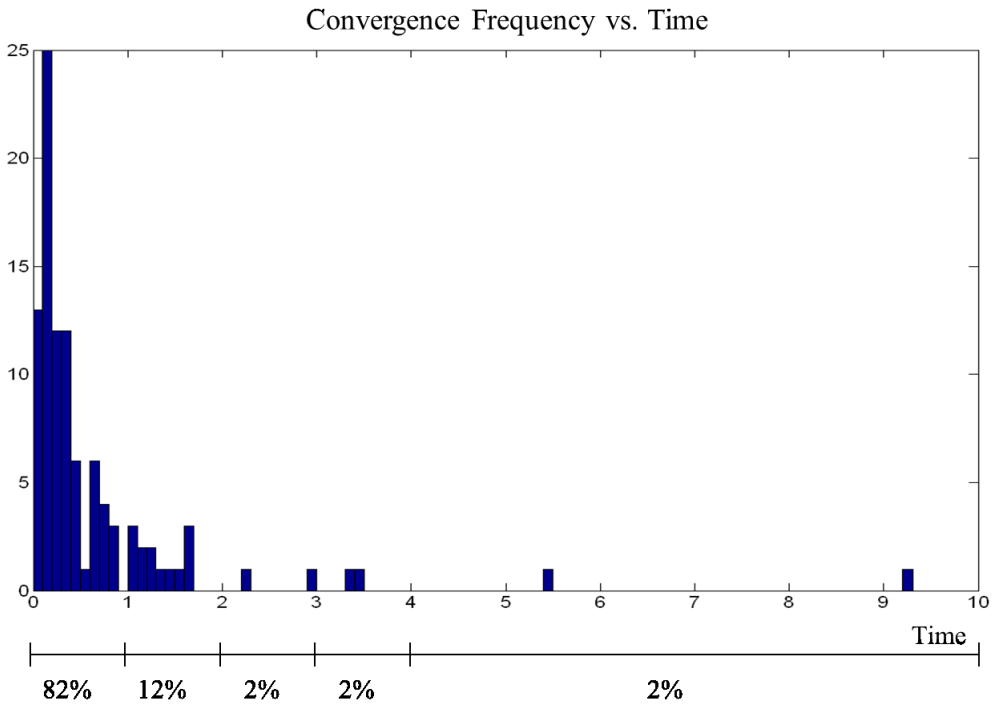


Figure 4.3: CPU time until convergence

In addition to analyzing the distributions of the performance measures, we also compare the performance measures when the optimal solution is found versus at termination. 82% of the problems are solved within one second. 62% of the problems are solved with five or fewer cuts. For 86% of the problems, the optimal solution is found with five or fewer cuts. When redundant cuts are removed from the problems, at termination there are five or fewer number of cuts for 80% of the problems. For the number of pivots, 57% of the problems are solved within 50 pivots, while this ratio is 81% to find the optimal solution. 85% of the problems are solved within 1000 solution evaluations. This ratio is 96% for 1000 or fewer solution evaluations to find the optimal solution. From these results, we conclude that most of the problems are solved within a small number of iterations.

The main conclusion from these observations is that time to find the optimal solution is a small portion of the total time. To validate these results, we continue the experiments over larger size test problems. However, before we give the results of these experiments, we describe the solver version improvements with the code changes. The version used so far is the initial prototype version in MATLAB. There are two inefficiencies of this version. The first one is the programming inefficiencies and the second one is the overhead time of MATLAB. To address the code inefficiencies, we rewrite the solver in MATLAB since MATLAB provides an easy experimentation framework. To address the overhead time of MATLAB, we program a version in FORTRAN. We expect the version in FORTRAN to have comparable times with commercial solvers to a certain extent. Table 4.3 gives the CPU time comparisons between different versions of our solver and LINDO.

Table 4.3: CPU time comparisons for different versions of solvers with LINDO

	MATLAB Initial	MATLAB Enhanced	FORTRAN Optimized	LINDO
Avg.	0.68	0.33	0.03	0.16
Max.	10.33	4.70	0.48	0.70

The results given in Table 4.3 show the importance of code improvements and programming language used. For the rest of this section, for prototyping purposes we use the enhanced version in MATLAB, he but for performance measuring purposes we use optimized version in FORTRAN.

We used $n = 30$ and $m = 15$ for all the experiments we have done so far. This is due to the slow convergence of the solver with the larger size problems and this behavior violates the quick experimentation requirement of the prototyping experiments.

Since we have determined the most efficient setting of the algorithm and have an optimized version of the solver, we continue with analyzing the experimental results over the test problems of larger size. We compare time to converge and time to find the optimal solution. For this comparison, we use all parameter combinations of the uncorrelated test suite. As a side note, since we are still in the probing stage in our experimentation, we prefer easier problems for quick experimentation. So we continue using the uncorrelated test suite.

Table 4.4 shows the comparison between the time until the optimal solution is found and the time to converge for the uncorrelated test suite of different size problems. These test runs use the optimized solver version in FORTRAN. The first header in the table is $n \times m$, where n is the number of variables and m is the number of constraints. We state median, maximum, and 80^{th} percentile of CPU time to have a better idea about the distribution of the values. The

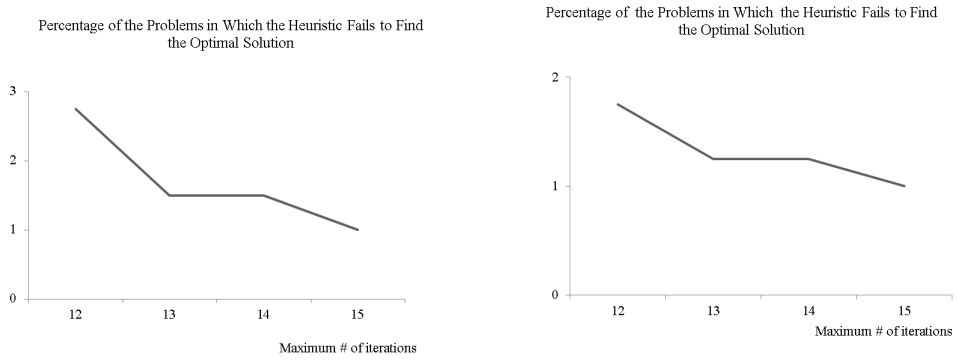
ratio of the time to find optimal solution to total time to converge is decreasing as the size of the problems is increasing. In other words, the optimal solution is found very quickly and the time to prove optimality becomes a high portion of the solve time. We use this conclusion as a motivation to investigate using this solver as heuristic.

Table 4.4: Time to find optimal solution vs time to converge for uncorrelated test suite

	30x15		40x20		50x25		60x30		70x35	
	Until opt.	Total	Until opt.	Total	Until opt.	Total	Until opt.	Total	Until opt.	Total
Avg.	0.02	0.03	0.04	0.20	0.12	1.83	0.40	12.19	0.88	51.65
Med.	0.01	0.02	0.02	0.07	0.06	0.33	0.15	1.43	0.33	6.42
Max	0.09	0.46	0.54	1.89	0.90	53.54	10.32	139.90	21.06	628.45
80 %	0.02	0.05	0.06	0.21	0.17	1.71	0.38	12.73	1.15	58.46

4.2.5 Heuristic Approach

In this section, we investigate the possibility of using the cutting plane solver as a heuristic by terminating the solver early. The first heuristic approach is to stop after a certain number of iterations. The second approach is to stop when the complexity of the separation problem is above a set parameter. We use the uncorrelated test problems for which $(n, m) \in \{(30, 15), (40, 20), (50, 25), (70, 35)\}$ for these experiments. Figure 4.4 shows the percentage of non-convergent problems versus different number of iterations for the pure cutting plane solver and the solver with n-way branching. The percentage of the non-convergent cases is very small for both versions of the solver.



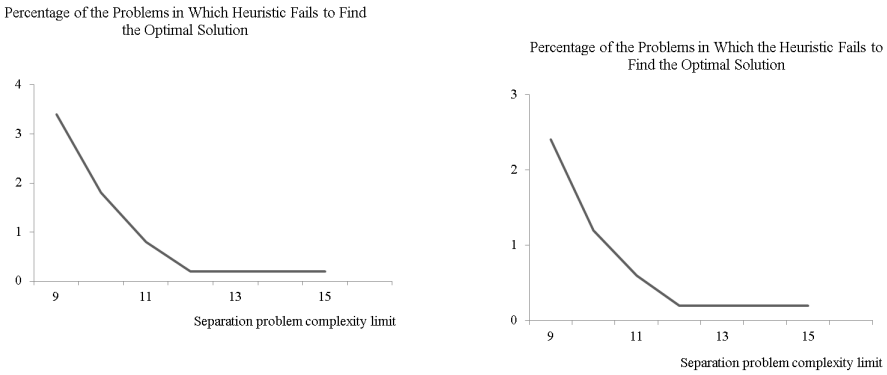
(a) Pure cutting plane solver

(b) Cutting plane solver with n-way branching

Figure 4.4: Failed-to-find-optimal-solution percentage for Heuristic 1

Figure 4.5 shows the percentage of the non-convergent problems if the solver is stopped when a certain complexity level is reached for the separation problem. We again see the same pattern as the first heuristic. The success rate is much higher with n-way branching compared to the pure cutting plane solver.

Setting a limit on the number of iterations is a more deterministic approach than setting a limit on complexity of the separation problem. However, setting a limit on complexity of the separation problem has a better chance of providing a higher quality solution and not spending more than the expected amount of time for cut generation. These results show that our approach has promising potential to be used as a heuristic.



(a) Pure cutting plane solver

(b) Cutting plane solver with n-way branching

Figure 4.5: Failed-to-find-optimal-solution percentage for Heuristic 2

4.2.6 Changes in the Experimentation Setting

For the rest of the experiments, we make three fundamental changes in our experiments. The main reason of these changes is the significant performance improvement with depth-k canonical cuts for $k = 2, 3$ which we report in the next section. This significant performance improvement motivates us to move toward a more sophisticated experimentation. The first change is that we extend our experiments to the correlated test suites. If the time to converge is short, machine noise might be high portion of the total time. To decrease the percentage of the machine noise over the total time, we include the correlated test suites. Generally, solvers take more time to converge while solving problems with the correlated coefficients than with the uncorrelated coefficients. Table 4.5, shows the performance measure values for the uncorrelated versus the correlated test suites for $n = 30$ and $m = 15$. Increases in the number of cuts generated and the time to converge are indications of the increase in the difficulty of the problems. There is a significant increase for time to converge. Having this test suite as part of our testing helps us see the performance changes easily.

Table 4.5: Performance measures for the correlated-easy test and the uncorrelated problems

	Correlated-easy			Uncorrelated		
	Cuts		Time	Cuts		Time
	Total	Until optimal	Total	Total	Until optimal	Total
Avg.	47.2	10.7	12.5	8.1	1.6	0.5
Max	100	49	39.7	44	7	8

From the initial experimental results, we conclude that as we add more features such as preprocessing and branching, there is significant improvement in the performance of the solver. This supports the idea that having a sequential solving methodology without branching or without preprocessing is not as effective as having these features on the solver. There has been wide research about the characteristics of a good solver. One of the most recent studies one is by Achterberg [1]. In this study, Achterberg gives some results on an effective way of using different features on SCIP solver, which is a constraint integer programming solver. Since this is a very wide research area, we proceed with our study by focusing more on the effectiveness of the canonical cuts.

The third change in the experimentation setting is to change the base solver. For the last probing experiment, we use only the pure cutting plane solver with the complexity limit of 15. The reason is to be able to analyze the changes in the canonical cut implementation independent from the other features. We continue with the last probing experiment and its results in the next section.

4.2.7 Various Depth Level Canonical Cuts

This section focuses on the experiments on various depth level canonical cuts within the enhanced cutting plane solver in MATLAB. The percentage of the volume cut is an indication of the quality of the cut and it is possible to increase the amount of the feasible region cut by increasing the depth level of the canonical cut. However increasing the depth level of the canonical cut increases the complexity of the separation problem. Figures 4.6 and 4.7 show the change in the percentage of the volume cut and separation problem complexity with changing depth level respectively.

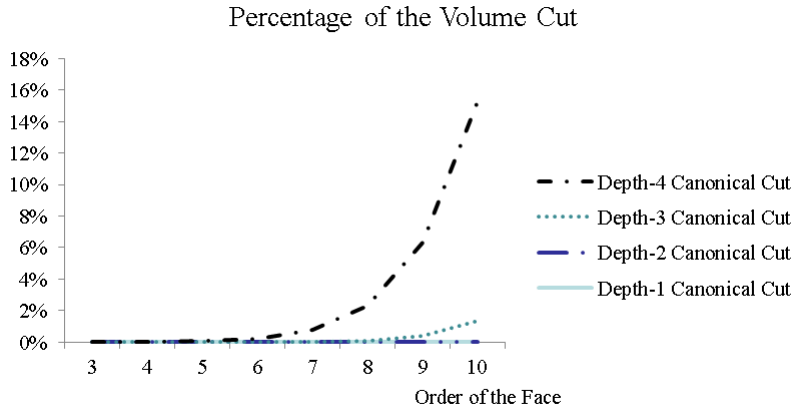


Figure 4.6: Percentage of the feasible region cut with various depth level cuts

We use the upper bound on the complexity of the separation problem and the upper bound on the percentage cut of the feasible region to show the consistency between the quality of the cuts and the complexity of the separation problems. The exponential nature of the complexity can be seen from these figures. To determine the actual trade off between various depth levels and quality of the cut, we do preliminary experiments using depth-1, depth-2, and depth-3 canonical cuts. The first part of this subsection states the results of an introductory experiment for the solver version with the depth-2 canonical cuts, which is generated using the face of order zero which contains the incumbent solution. The second subsection reports results on comparison of depth-1, depth-2, and depth-3 canonical cuts. In these experiments, we do not go beyond depth-3 canonical cuts because of the performance drop after depth-2 canonical cuts. In other words, the trade off between time and quality passes the break-even point after depth-2 canonical cuts. To evaluate the performance changes, we use uncorrelated and correlated problems, and since these are introductory experiments, we keep the problem size small at

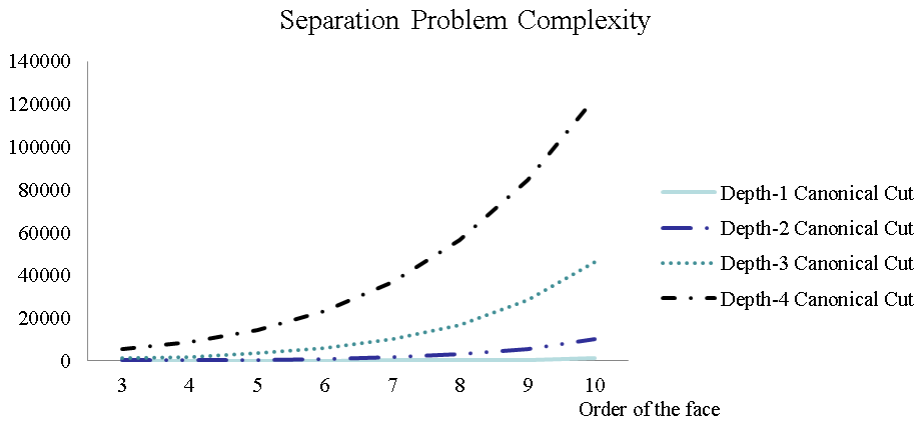


Figure 4.7: Separation problem complexity with various depth level canonical cuts

$n = 30$ and $m = 15$.

Depth-2 Canonical Cuts from F^0

The increase in the complexity of the separation problem from depth-1 to depth-k canonical cuts, for $k > 1$, is the main reason we have not implemented these cuts in the initial experiments. However, if the order of the face the cut is generated from is small, the separation problem is manageable. The order of the face that contains an incumbent solution is 0 since a feasible solution is a vertex. The complexity of the separation problem for depth-2 cut for any face with order 0 is $O(n)$. Tables 4.6 and 4.7 give the comparison of some point statistics between the solver with and without depth-2 canonical cut at the incumbent solution. Improved statistics are identified in bold fonts.

Table 4.6: Statistics for correlated-easy test suite with depth-2 cut at incumbent solution

	W/O D2 at incumbent				W/ D2 at incumbent			
	Mean	Median	Max.	80%	Mean	Median	Max.	80%
# of cuts	47.2	44	100	66	**46.7**	**42.5**	100	66
# of cuts until optimal	10.7	5	49	21	**10.4**	5	49	**15**
# of nonredundant cuts	35.1	33	70	50	39.7	39.5	81	52
CPU time	12.5	8.2	39.7	22	15.4	10.2	51	26.6

For correlated-easy problems, the average time to converge increase while the average number of cuts generated decreased. Therefore, over the correlated-easy test suite, this imple-

Table 4.7: Statistics for uncorrelated test suite with depth-2 cut at incumbent solution

	W/O D2 at incumbent				W/ D2 at incumbent			
	Mean	Median	Max.	80%	Mean	Median	Max.	80%
# of cuts	8.1	4	44	12	8.3	5	**36**	16
# of cuts until optimal	1.6	1	7	3	1.8	**0**	18	2
# of nonredundant cuts	5.8	7	27	8	9	**6.5**	30	13
CPU time	0.5	0.3	8	0.4	0.7	**0.1**	**7**	0.6

mentation does not help. For uncorrelated test suite, depth-2 canonical cut median time and maximum time decreased. Overall, adding this cut over depth-1 cut does not seem to improve the results.

Depth-1, Depth-2 and Depth-3 Canonical Cuts

When depth-2 canonical cut is generated only at incumbent solution in addition to depth-1 canonical cut, the performance does not improve very much. However, due to Corollary 2.4.4, depth-2 canonical cut has the potential to cut more region than the region cut by equivalent set of depth-1 cuts. We can extend this conclusion to a deeper level of cuts. For example, depth-3 cut is stronger than the equivalent set of depth-2 or depth-1 cuts. In other words, in theory depth-3 canonical cuts are stronger than depth-1 and depth-2 canonical cuts. However it does not necessarily mean that it provides the shortest time to converge. In this subsection, we compare the performance of depth-1, depth-2, and depth-3 canonical cuts. Our purpose is to find the trade-off between quality and complexity among canonical cuts at different depth levels.

Figures 4.8 and 4.9 show the performance profile curves for time to converge. The procedure to plot performance profiles is adapted from [14]. The x-axis is the log of the normalized time, $log_2(r)$, over different solver versions. The y-axis is the percentage of the problems solved within the corresponding $log_2(r)$. We give more information on performance profile curves in Subsection 4.3.2. Table 4.8 gives the legend description of these figures.

Table 4.8: Solver version descriptions

Version Acronym	Description
WD1	Solver with *depth 1* canonical cuts
WD1+	Solver with *depth 1* and *depth 2*-at-incumbent canonical cuts
WD2	Solver with *depth 2* canonical cuts
WD3	Solver with *depth 3* canonical cuts

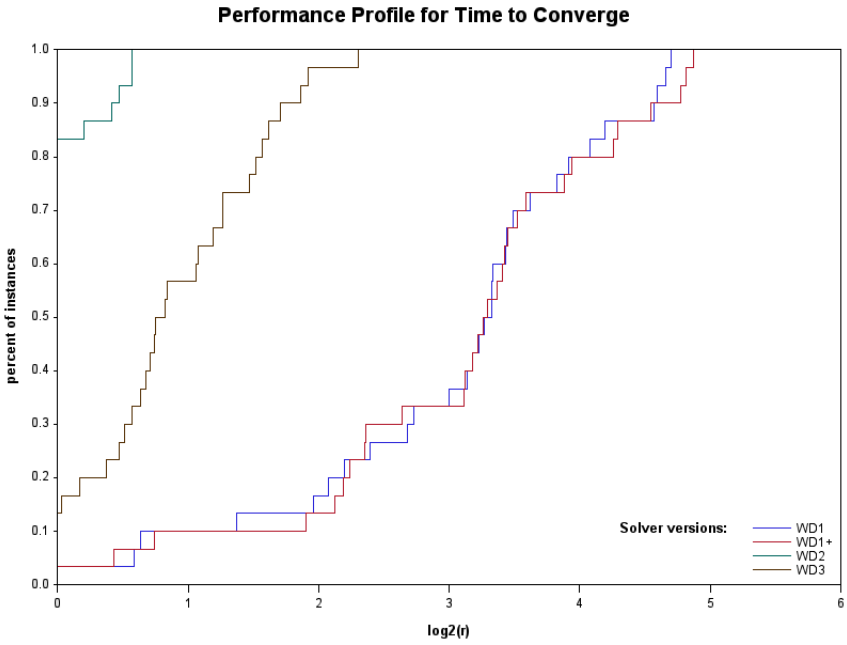


Figure 4.8: Performance profile for time to converge for correlated-easy problems

The best performing solver version over both test suites is the one with depth-2 canonical cuts. However, the order of the rest of the solver versions is different. For correlated-easy problems, WD3 dominates WD1 and WD1+, while for uncorrelated problems, WD3 performs poorly. Generally, uncorrelated problems take less time than correlated problems. They can be considered as easy problems. WD3 provides the cut with the most quality. However for easy problems, the benefit from WD3 does not compensate for the time it requires. From this experimental result, we conclude that depth-2 canonical cuts provide the best result. This conclusion determines the design of experiments with the SAS/OR MILP solver.

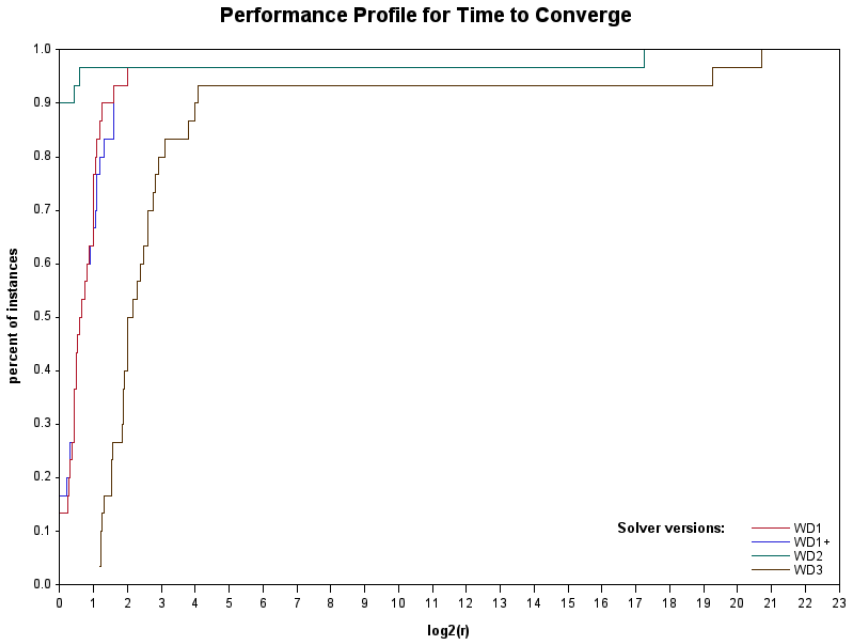


Figure 4.9: Performance profile for time to converge for uncorrelated problems

4.2.8 Conclusion of the Preliminary Experiments

As already mentioned, the experiments presented so far are preliminary experiments to determine the direction of our study. From these experiments, we conclude that there are two features that make significant improvements. The first one is the n-way branching. The second one is the various depth level canonical cuts. Canonical cuts are applicable to both pure cutting plane and branching algorithms. In addition to this, to the best of our knowledge, this is the first time various depth level canonical cuts are studied. So we decide to focus on canonical cuts for the rest of our study.

4.3 Experimental Results of Canonical Cuts within SAS/OR MILP solver

In this section, we analyze the experimental results of the different implementations of the canonical cuts within the SAS/OR MILP solver over OR Library problems. This section has two types of experiments. In the first part, we measure the significance of the proposed implementations using Wilcoxon signed rank test over a variety of randomly generate correlated problems. In the second part, we use performance profiles for measuring the performance changes over OR Library problems, which are used for benchmarking purposes in literature [11].

4.3.1 Effects of the Proposed Implementations on SAS/OR MILP Solver over Variety of Problems

The main purpose of this section is to measure the effectiveness of different implementations over a variety of test problems and detect when implementations start failing. To measure the effectiveness of the implementations with different parameters, we use the Wilcoxon signed rank test. The test problems are generated using the correlated test problem generator described in Section 4.1. However, due to the performance differences of the different components of the SAS/OR MILP solver and our canonical cut implementations, performance measures which are time to converge and relative gap do not have a normal distribution. Due to this reason, we choose the Wilcoxon signed rank test to do a pairwise performance comparison of proposed implementations with the solver itself. We vary the number of variables, n, the number of constraints, m, and the tightness ratio, α. We limit ourself to three levels for each factor due to the rapid increase in the number of the possible combinations. We experiment on the full factorial combinations, in other words 27 combinations. For each combination we generate 20 problems. The following list states the values of each parameter.

- $n \in \{50, 100, 150\}$
- $m \in \{15, 25, 50\}$
- $\alpha \in \{0.15, 0.25, .50\}$

We compare the performance of the three implementations added onto the SAS/OR MILP solver with the solver itself. With the first implementation, we add only depth-1 canonical cuts. The cut is turned off if the current solution is on a face of which the order is above a limit (lim). This implementation is referred to as WD1_lim. The second implementation is very similar to the first implementation. The only difference is adding depth-2 canonical cut instead of depth-1 canonical cut. The second implementation is referred to as WD2_lim. For both of

these implementations, $lim \in \{10, 20, 30, 40\}$. The reason to limit this parameter to 40 is the inefficiency of implicit enumeration for solving the separation problem when the complexity of the separation problem is above $O(2^{40})$. The third implementation is adding either depth-1 or depth-2 cut depending on the order of the face the current solution is on. Depth-2 cut is the default cut. This cut is on as long as the order of the current face is less than or equal to $\lim_2$. If the order of the face is above $\lim_2$, then depth-2 cut is turned off. If depth-2 cut is off and the order of the face is less than $\lim_1$, then depth-1 cut is on. If the order of the face is greater than $\lim_1$, then neither depth-1 or depth-2 cut is on. This implementation is referred to as the *hybrid* implementation and the acronym we use for this implementation is HYB_$\lim_1$_$\lim_2$ such that $(\lim_1, \lim_2) \in \{(10, 5), (20, 15), (30, 25), (40, 35)\}$.

There are multiple aspects that affect the performance. The first one is whether the total time for solving separation problems is compensated by the improvement gained by the canonical cuts. The second one is the deviation of the performance of the solver's features such as different cuts and heuristics on different types of problem. The last one is the randomness of the problems. In other words, the variation of the difficulty of the problems might prevent the results from being consistent. We would like to measure the effect of the first aspect. However, since there are other things that affect the performance, this makes performance changes with the proposed implementations difficult to measure.

We use either time to converge, relative gap, or both to compare the performance. If the solver converged for all of the test problems, then we compare only time to converge. If the solver reached the time limit for all the problems, then we use relative gap to measure the performance. If the solver converged for some of the problems while it reached the time limit for the rest of the problems, we use both measures. We limit the time to 15 minutes due to resource limitations.

We report the p-values of the Wilcoxon signed rank test in Tables 4.9 and 4.12for $n = 50$, in Tables 4.15, 4.16 and 4.17 for $n = 100$ and in Tables 4.18, 4.19 and 4.20 for $n = 150$. p-values represent the significance of the difference between the solver without any implementations and the /;solver with each implementation with each parameter. Any p value less than 0.05 implies the superiority of the implementation with the specific parameter while p values close to 1 imply the superiority of the solver itself. Any p value in between implies that our results are not enough to conclude superiority of the implementation or the solver by itself. We use bold letters for the superior or inferior implementations.

When the convergence is fast, the effect of the implementations is more significant. The convergence of the solver is faster for $n = 50$ compared to other n values and for most of the implementations over the test problems, so results have been significantly improved by the proposed implementations. For $n = 50$, we analyze both relative gap and time to converge. Depth-2 cut with parameter greater than or equal to 30 becomes very inefficient for cases of

Table 4.9: P-values of Wilcoxon signed rank test for time to converge for $n = 50$ and $m = 15$

Implementation 1				
α	WD1_10	WD1_20	WD1_30	WD1_40
0.15	0.0689	**0.024**	**<0.0001**	**0.0006**
0.30	0.6603	**0.0001**	**<0.0001**	**<0.0001**
0.50	0.6914	**0.0073**	**0.0059**	**0.0059**
Implementation 2				
α	WD2_10	WD2_20	WD2_30	WD2_40
0.15	0.3072	**<0.0001**	0.9621	0.9587
0.30	0.9022	**<0.0001**	**0.007**	**0.007**
0.50	0.394	**0.0461**	**0.0241**	**0.0132**
Implementation 3				
α	HYB_10_5	HYB_20_15	HYB_30_25	HYB_40_35
0.15	**0.0002**	**<0.0001**	0.8464	0.9587
0.30	**0.2119**	**<0.0001**	**0.0003**	**0.007**
0.50	**0.2053**	0.1162	**0.0172**	**0.0282**

Table 4.10: P-values of Wilcoxon signed rank test for time to converge for $n = 50$ and $m = 25$

Implementation 1				
α	WD1_10	WD1_20	WD1_30	WD1_40
0.15	0.8593	**<0.0001**	**<0.0001**	**<0.0001**
0.30	**0.0353**	0.8685	**<0.0001**	**0.0002**
0.50	0.1719	0.543	0.4573	0.7451
Implementation 2				
α	WD2_10	WD2_20	WD2_30	WD2_40
0.15	0.2697	**<0.0001**	**<0.0001**	**1**
0.30	0.5958	0.6612	**0.9999**	**1**
0.50	**0.042**	0.0723	**0.9999**	**1**
Implementation 3				
α	HYB_10_5	HYB_20_15	HYB_30_25	HYB_40_35
0.15	**<0.0001**	**<0.0001**	**<0.0001**	**<0.0001**
0.30	0.0968	0.087	**<0.0001**	**0.9999**
0.50	0.1953	0.4805	**0.999**	**1**

Table 4.11: P-values of Wilcoxon signed rank test for time to converge for $n = 50$ and $m = 50$

Implementation 1				
α	WD1_10	WD1_20	WD1_30	WD1_40
0.15	N/A	N/A	N/A	N/A
0.30	N/A	N/A	N/A	N/A
0.50	N/A	N/A	N/A	N/A
Implementation 2				
α	WD2_10	WD2_20	WD2_30	WD2_40
0.15	N/A	N/A	N/A	N/A
0.30	N/A	N/A	N/A	N/A
0.50	N/A	N/A	N/A	N/A
Implementation 3				
α	HYB_10_5	HYB_20_15	HYB_30_25	HYB_40_35
0.15	N/A	N/A	N/A	N/A
0.30	N/A	N/A	N/A	N/A
0.50	N/A	N/A	N/A	N/A

Table 4.12: P-values of Wilcoxon signed rank test for relative gap for $n = 50$ and $m = 15$

Implementation 1				
α	WD1_10	WD1_20	WD1_30	WD1_40
0.15	N/A	N/A	N/A	N/A
0.30	N/A	N/A	N/A	N/A
0.50	0.4039	0.1384	0.1514	0.1514
Implementation 2				
α	WD2_10	WD2_20	WD2_30	WD2_40
0.15	N/A	N/A	N/A	N/A
0.30	N/A	N/A	N/A	N/A
0.50	0.3632	**<0.0001**	**<0.0001**	**<0.0001**
Implementation 3				
α	HYB_10_5	HYB_20_15	HYB_30_25	HYB_40_35
0.15	N/A	N/A	N/A	N/A
0.30	N/A	N/A	N/A	N/A
0.50	**0.0101**	**0.0001**	**0.0001**	**0.0001**

Table 4.13: P-values of Wilcoxon signed rank test for relative gap for $n = 50$ and $m = 25$

Implementation 1				
α	WD1_10	WD1_20	WD1_30	WD1_40
0.15	N/A	N/A	N/A	N/A
0.30	0.8887	0.1011	**0.0007**	**0.0017**
0.50	**0.0253**	0.1193	**<0.0001**	**<0.0001**
Implementation 2				
α	WD2_10	WD2_20	WD2_30	WD2_40
0.15	N/A	N/A	N/A	N/A
0.30	0.626	0.4854	**1**	**1**
0.50	**0.0454**	**0.0101**	**1**	**1**
Implementation 3				
α	HYB_10_5	HYB_20_15	HYB_30_25	HYB_40_35
0.15	N/A	N/A	N/A	N/A
0.30	**0.0127**	**0.0131**	**0.0028**	**1**
0.50	**0.0052**	**0.0054**	**0.9999**	**1**

Table 4.14: P-values of Wilcoxon signed rank test for relative gap for $n = 50$ and $m = 25$

Implementation 1				
α	WD1_10	WD1_20	WD1_30	WD1_40
0.15	**<0.0001**	**<0.0001**	**<0.0001**	**0.0001**
0.30	**<0.0001**	**<0.0001**	**<0.0001**	**0.0001**
0.50	**<0.0001**	**<0.0001**	**<0.0001**	**0.0001**
Implementation 2				
α	WD2_10	WD2_20	WD2_30	WD2_40
0.15	**<0.0001**	**<0.0001**	**1**	**1**
0.30	**<0.0001**	**<0.0001**	**1**	**1**
0.50	**<0.0001**	**<0.0001**	**1**	**1**
Implementation 3				
α	HYB_10_5	HYB_20_15	HYB_30_25	HYB_40_35
0.15	**<0.0001**	**<0.0001**	**1**	**1**
0.30	**<0.0001**	**<0.0001**	**1**	**1**
0.50	**<0.0001**	**<0.0001**	**1**	**1**

Table 4.15: P-values of Wilcoxon signed rank test for relative gap for $n = 100$ and $m = 15$

Implementation 1				
α	WD1_10	WD1_20	WD1_30	WD1_40
0.15	0.8066	0.209	0.1868	0.1868
0.30	0.4673	0.8441	0.8441	0.8419
0.50	**0.0007**	**0.0029**	**0.0042**	**0.919**
Implementation 2				
α	WD2_10	WD2_20	WD2_30	WD2_40
0.15	0.7812	**0.0007**	**0.0028**	**0.0024**
0.30	0.156	**0.0413**	**0.0348**	**0.0348**
0.50	0.5929	**0.0021**	**0.0032**	**0.0032**
Implementation 3				
α	HYB_10_5	HYB_20_15	HYB_30_25	HYB_40_35
0.15	**0.0292**	**0.0003**	**0.0047**	**0.0021**
0.30	**0.0117**	0.1559	**0.0319**	**0.0319**
0.50	**0.0186**	0.0615	**0.032**	**0.0024**

Table 4.16: P-values of Wilcoxon signed rank test for relative gap for $n = 100$ and $m = 25$

Implementation 1				
α	WD1_10	WD1_20	WD1_30	WD1_40
0.15	0.1407	0.2608	**0.0006**	**0.0001**
0.30	0.336	0.1855	0.0086	0.0086
0.50	0.065	**0.0435**	0.1387	0.1227
Implementation 2				
α	WD2_10	WD2_20	WD2_30	WD2_40
0.15	**0.0204**	**0.0125**	**1**	**1**
0.30	0.4648	0.2855	**1**	**1**
0.50	**0.0045**	**0.0017**	**1**	**1**
Implementation 3				
α	HYB_10_5	HYB_20_15	HYB_30_25	HYB_40_35
0.15	**0.0311**	**0.0028**	**0.0068**	**1**
0.30	**0.0311**	**0.0284**	**1**	**1**
0.50	**0.0003**	**0.0001**	**0.994**	**1**

Table 4.17: P-values of Wilcoxon signed rank test for relative gap for $n = 100$ and $m = 50$

Implementation 1				
α	WD1_10	WD1_20	WD1_30	WD1_40
0.15	**0.0007**	**0.0029**	**0.0042**	**0.919**
0.30	0.2759	0.3289	0.2543	0.3011
0.50	0.4097	0.4568	0.4167	0.5957
Implementation 2				
α	WD2_10	WD2_20	WD2_30	WD2_40
0.15	**0.0003**	**0.0125**	**0.0024**	**0.9291**
0.30	0.3008	0.489	0.686	0.4029
0.50	0.1285	**0.031**	0.0806	0.0632
Implementation 3				
α	HYB_10_5	HYB_20_15	HYB_30_25	HYB_40_35
0.15	**0.0002**	**<0.0001**	**<0.0001**	**<0.0001**
0.30	**<0.0001**	**0.0075**	**0.0116**	**0.0088**
0.50	0.0004	**<0.0001**	**<0.0001**	**<0.0001**

Table 4.18: P-values of Wilcoxon signed rank test for relative gap for $n = 150$ and $m = 15$

Implementation 1				
α	WD1_10	WD1_20	WD1_30	WD1_40
0.15	0.3802	0.3506	0.4347	0.3643
0.30	0.1498	**0.0053**	**0.0053**	**0.0053**
0.50	0.4953	0.1942	0.1942	0.1942
Implementation 2				
α	WD2_10	WD2_20	WD2_30	WD2_40
0.15	0.4862	0.3108	0.1081	0.1081
0.30	0.0916	**0.0086**	**0.0086**	**0.0086**
0.50	0.0912	0.4598	0.4636	0.4636
Implementation 3				
α	HYB_10_5	HYB_20_15	HYB_30_25	HYB_40_35
0.15	**0.0294**	0.165	0.0715	0.1081
0.30	**0.0358**	**0.0016**	**0.0077**	**0.0077**
0.50	**0.0392**	**0.006**	0.4026	0.4062

Table 4.19: P-values of Wilcoxon signed rank test for relative gap for $n = 150$ and $m = 25$

Implementation 1				
α	WD1_10	WD1_20	WD1_30	WD1_40
0.15	**0.0005**	**0.0004**	**0.0096**	**0.0096**
0.30	0.0865	0.0596	0.1012	0.0996
0.50	**0.0168**	**0.0094**	0.7238	0.7021
Implementation 2				
α	WD2_10	WD2_20	WD2_30	WD2_40
0.15	**0.0147**	**0.0156**	**0.9947**	**0.9947**
0.30	0.0655	0.0808	**0.9996**	**0.9996**
0.50	0.3354	0.2117	**0.9996**	**0.9996**
Implementation 3				
α	HYB_10_5	HYB_20_15	HYB_30_25	HYB_40_35
0.15	**0.0002**	**0.0046**	**0.9904**	**0.9947**
0.30	**0.0052**	**0.0059**	**0.9996**	**0.9996**
0.50	**<0.0001**	0.0631	**1**	**0.996**

Table 4.20: P-values of Wilcoxon signed rank test for relative gap for $n = 150$ and $m = 50$

Implementation 1				
α	WD1_10	WD1_20	WD1_30	WD1_40
0.15	0.093	0.2789	0.1134	0.1001
0.30	0.3952	0.4957	0.5801	0.5403
0.50	**0.0276**	**0.0182**	**0.0145**	**0.0243**
Implementation 2				
α	WD2_10	WD2_20	WD2_30	WD2_40
0.15	0.1028	0.1672	0.1226	0.1283
0.30	0.7853	0.5044	0.5847	0.7395
0.50	**0.024**	**0.0137**	0.103	**0.0067**
Implementation 3				
α	HYB_10_5	HYB_20_15	HYB_30_25	HYB_40_35
0.15	0.086	0.0962	**0.0355**	**0.0311**
0.30	**0.0357**	0.3386	0.3501	0.1629
0.50	**0.002**	**0.0005**	**0.0016**	**0.0024**

$m = 25$ and $m = 50$. This implies that the time spent to solve the separation problem is not compensated by the improvement gained from depth-2 cut. For these test problems, the higher number of constraints in addition to the larger feasible region increases the potential of solutions on high order faces, and this increases the solving time of the separation problem.

For $n = 100$, the most effective implementation is the hybrid implementation. For the hybrid implementation for every region size, there is at least one parameter that improves the performance of the solver. Some implementations with depth-2 cuts become very inefficient when the limit parameter is above 25. We see this behavior when the number of constraints is 25. Solver versions WD2_30, WD2_40, and HYB_40_35 show this behavior for all α values. HYB_30_25 shows the same behavior for all α values except 0.15. With the limit parameter over 25 with depth-2 cut, solving the separation problem becomes very expensive. Solver versions with hybrid implementations consistently perform better than the solver itself for $m = 50$ with all parameters.

For $n = 150$, the hybrid implementation performs significantly better than the first two implementations. For each region size, there exists at least one parameter that outperforms the solver itself.

Overall, parameter values close to the number of constraints have the potential to work better. For example, for $n = 150$, when $m = 15$, $lim > 15$ is preferred as long the cuts are affordable. If these cuts are not affordable, it is possible to gain some benefit from a smaller limit parameter as well. In some of the implementations, we can observe the trade off between the time and the quality of the cuts from the change of p-values. For example for $n = 150$, over test problems of $m = 25$ and $\alpha = 0.50$, first there is a decrease and then there is an increase in p-value as we increase the limit parameter of the first implementation. The lowest p-value gives the preferred parameter for each implementation.

4.3.2 Performance Changes over OR Library Problems

In this part of the experimental results we analyze the performance changes over OR Library problems with proposed implementations. These problems are first used by Chu and Beasley [11]. Other researches use these test problems for benchmarking purposes. In the first part of this section, we use performance profiles for measuring the performance changes with different implementations. In the second part, we compare the solution found with the best performing implementation with the best known solutions in literature. We were only able to find the best known solutions for $n = 500$. So our comparison will be limited to these problems.

Measuring the Performance Changes Using Performance Profiles

We used the performance profiles that are proposed by Dolan and More [14]. Performance profile curves are plotted using the probability distribution function for a performance measure. This tool is widely used by researchers for benchmarking and comparing optimization software. Let $p_{s,t}$ represent a performance measure for the solver version $s \in S$ for test problem $t \in T$ where S is the set of all solver versions and T is the set of all test problems. We assume that a smaller value of $p_{s,t}$ is an indication of good performance. This performance measure is normalized with the following formulation for a solver version $s' \in S$:

$$\overline{p}_{s',t} = \frac{p_{s',t}}{\min\limits_{s \in S} p_{s,t}}.$$

If the minimum of the performance measure among all the solver versions is zero, we replace this value with $1e^{-5}$ to avoid division by zero. The following equation shows the cumulative probability distribution function that is used to plot the performance profile curves:

$$F_s(\tau) = \frac{1}{|T|} \left| \{ t \in T : \overline{p}_{s,t} \leq \tau \} \right|$$

where $|T|$ indicates the cardinality of the set T. $F_s(\tau)$ is the probability estimate that the normalized performance measure for solver version s is within a factor of τ of the best performance measure. In other words, $F_s(\tau)$ gives the percentage of the test problems that solver version s solves given that the time limit is τ times the minimum time for each test problem among all the solver versions. We use the performance profile curves for time and relative gap comparisons. If the solver converges for all problems within a given time limit, we use time to converge to evaluate the performance. If the solver reaches the time limit for all problems, then we use relative gap to evaluate the performance changes. If the solver converges for some problems while reaching the time limit for some other, we use both to evaluate the results.

Each solver version is represented by one of the acronyms described in Section 4.3.1. As a reminder, implementations 1 and 2 with the turn-off parameter of lim are referred to as WD1_lim and WD2_lim, respectively; implementation 3 with turn-off parameters of lim_1 and lim_2 is referred to as HYB_lim_1_lim_2.

We get the most significant results when we classify the results with respect to the number of constraints, $m \in \{5, 10, 30\}$.

Performance profiles for time to converge and relative gap for test problems with $m = 5$ are given in Figures B.1, B.2 and B.3 in Appendix B for implementations 1, 2, and 3, respectively. The best performing parameter for $m = 5$ with the first and second implementations is when cut turn off parameter is set to 5. Other parameters degraded the performance of these imple-

mentations. All the implementations with all the parameters performed worse than the solver itself with hybrid implementation.

Performance profiles for time to converge and relative gap for test problems with $m = 10$ are given in Figures B.4, B.5 and B.6 in Appendix B for implementations 1, 2, and 3, respectively. For $m = 10$, WD1_5 performs very close to the solver itself. Implementation 1 with the limit parameter greater than five improved the results significantly. As we increase the parameter above 10, performance improvements get steady. When we look at the performance profile for relative gap, we see the same pattern. Having no more performance improvement after a certain parameter is mostly due to not having solutions on higher order faces very often. With the second implementation, the best performing parameter is 15 both for time to converge and relative gap. There is not much drop in the performance as we increase the limit parameter value, which is opposite of what we have seen in some cases explained in Section 4.3.1. This is again due to not having many LP solutions on higher order faces. As opposed to the first two implementations, the best parameter values are higher. The best performing solver versions are HYB_30_25 and HYB_35_30.

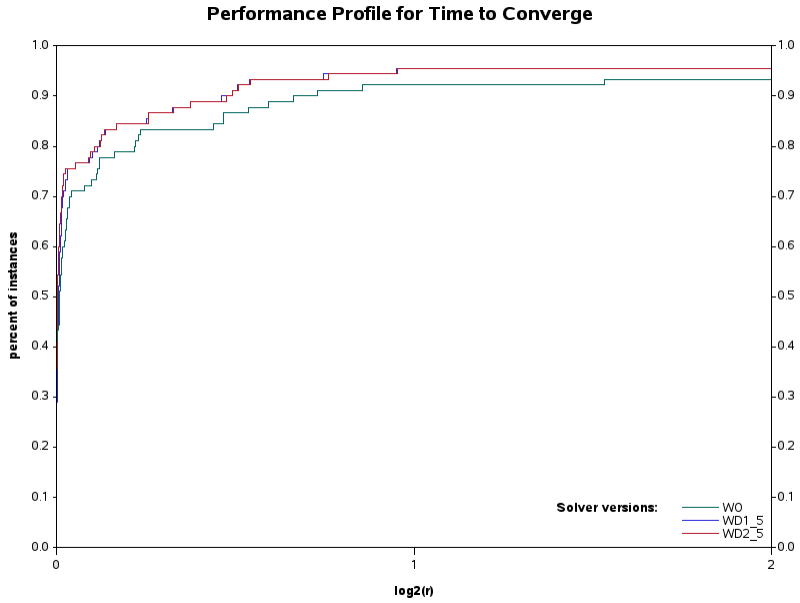
Performance profiles for time to converge and relative gap for test problems with $m = 30$ are given in Figures B.7, B.8 and B.9 for implementations 1, 2, and 3, respectively. For implementation 1, the best performing solver versions are WD1_25 and WD1_30. The performance drops when we increase the limit parameter beyond 25. If we look at the second implementation, WD2_20 is the best performing solver version for the cases where the solver converges. This version increases the convergence rate. When we compare the relative gap of the test problems for which the solver reaches the time limit, we see a slight increase in performance with solver version WD2_20. There are two solver versions of hybrid implementation that perform better than the solver itself. These are HYB_20_15 and HYB_25_20. The best performing one is HYB_25_20.

From the analysis above, we can conclude that parameter values of the proposed implementations are very important to improve the performance of the solver. If the limit parameter is close to the number of constraints, it is more effective. However, it is important to consider when a cut is not affordable due to expensive separation problem solving. Also, from these observations we can observe the trade off between time and quality of the cut and when this relationship reaches or passes the breakeven point.

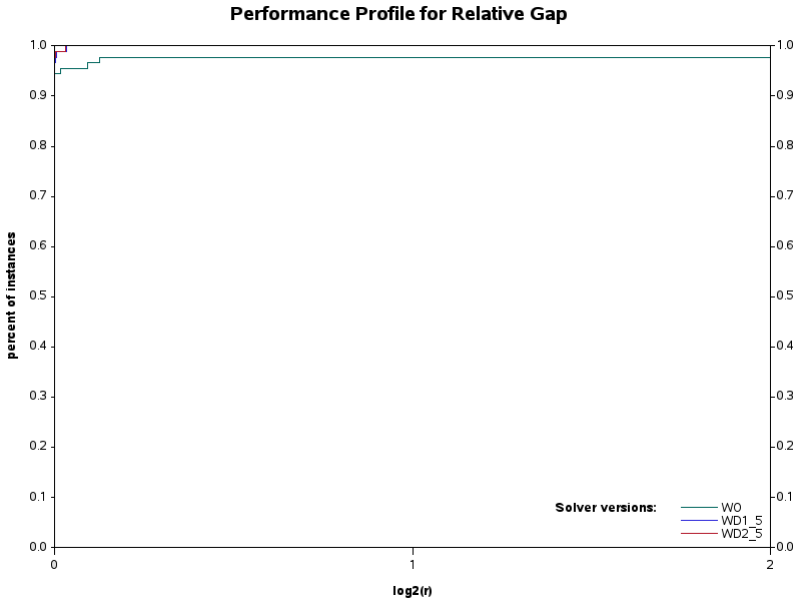
We continue this subsection by comparing the best performing solver versions of each implementation with each other. For $m = 5$, from our observations we choose WD1_5 and WD2_5. Figure 4.10 shows the performance profiles for the relative gap and the time to converge. As stated before, there is significant performance improvement with these versions for the solver. However, there is no significant difference between these solver versions both for the time to converge and the relative gap with each other.

Figure 4.11 shows the best performing solver versions of all implementations for test problems with $m = 10$. For time to converge, WD2_20 and HYB_25_20 improve the performance more than the other solver versions. Both of these versions increase the convergence rate as well. When we compare the solver versions using the relative gap for test problems that the solver failed to solve, all of the versions perform significantly better than the solver itself. Among the versions, WD2_20 and HYB_25_20 give slightly better results than the other versions.

Figure 4.12 shows the best performing solver versions of all implementations for test problems with $m = 30$. From this figure, we observe that the improvement of the solver is more significant. All the versions except WD2_20 perform very close to each other. However, the performance profile curve for HYB_25_20 is above the other curves more often. We observe the same type of behavior for relative gap as well.

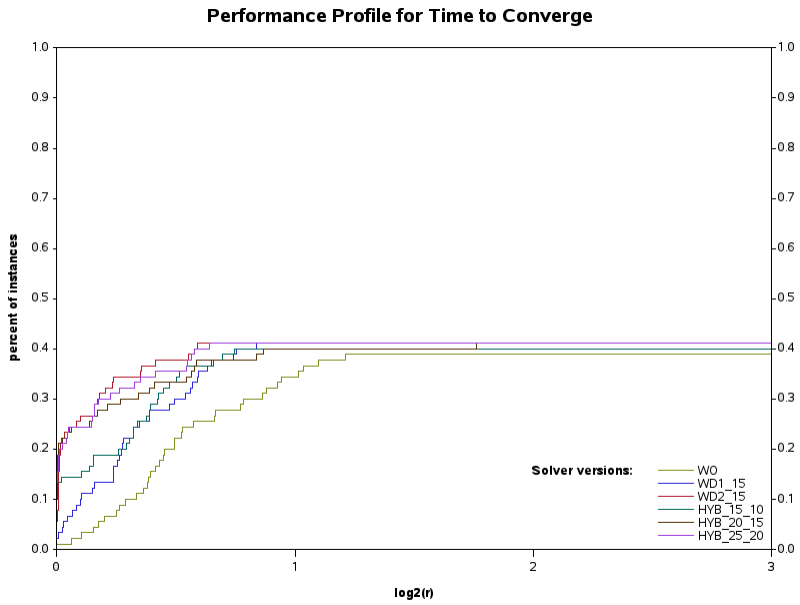


(a) Time to converge

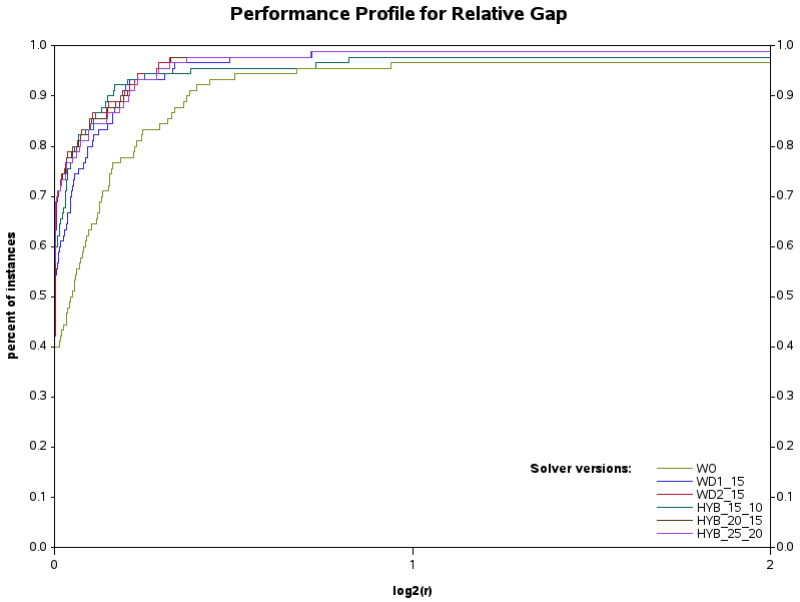


(b) Relative gap

Figure 4.10: Performance profiles for ORLIB test problems with $m = 5$ of the best solver versions

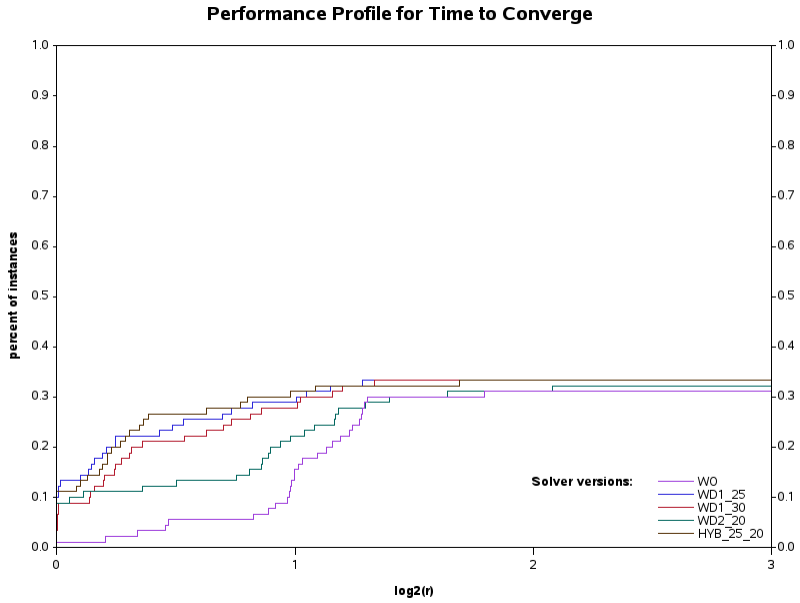


(a) Time to converge

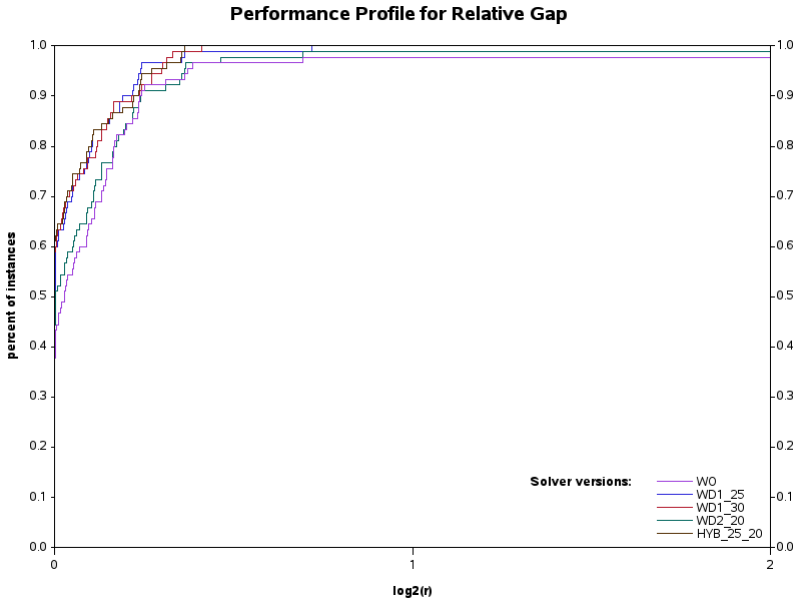


(b) Relative gap

Figure 4.11: Performance profiles for ORLIB test problems with $m = 10$ of the best solver versions



(a) Time to converge



(b) Relative gap

Figure 4.12: Performance profiles for ORLIB test problems with $m = 30$ of the best solver versions

Comparing the Solution Found with the Best Known Solution

In this section, we state the results that show the solution improvement of the SAS/OR solver with the proposed implementations. As mentioned before, the test problems considered for this analysis are the ones with $n = 500$. For $m = 5$, there is no improvement over the solver itself with the best performing implementations. The solver itself already finds the best known solution for most of the cases. There are five problems where the solver finds a better solution than the best known solution and there is a single case solver finds a worse solution than the best known solution. Overall, we can say for $m = 5$ the solver's performance is very good. For the cases with $m = 10$, over 30 problems, the best implementation improved the results for 21 of these problems. For the remaining problems, the solution found is the same as the one solver finds. There are two test cases where the solver with the best implementation finds a better solution than the best known, and for nine cases it finds the same solution as the best known. For the rest of the problems, the solver with the best implementation finds a worse solution than the best known solution. For $m = 30$, the solver solution is improved over 24 test problems. Except for one case, the solver with the best implementation finds worse solution than the best known solution. Tables of the comparisons are given in Appendix C.

4.3.3 Summary and Conclusions

In this subsection, we summarize our observations that are stated in Sections 4.3.1 and 4.3.2 and state important conclusions we have reach. Proposed implementations improve the solver performance significantly. Improvements depend on the parameter setting and the problem that is solved. In other words, we need to consider the problem specifications to determine the parameter of the implementations. One thing to keep in mind when determining the parameter for an implementation is the limitations. For example, solving the separation problem becomes very expensive when the solver starts encountering LP solutions on faces of order greater than or equal to 30 very often. In such a case, the time spent for solving the separation problem is not compensated by the improvements gained with the proposed implementations. This behavior is especially very obvious while solving problems with high number of constraints and large feasible regions with the implementations that have the depth-2 canonical cuts. In generic terms, our observations show that the proposed implementations improve the performance up to a certain parameter value, then the performance starts degrading.

The rule of thumb to determine the best performing parameter value is to be around the the number of constraints. When the solver has a hard time to solve a test problem, a higher parameter value performs better. However, this value should still be around the number of constraints. For example, the convergence rate of the solver is very high for OR Library problems with $m = 5$, and for these problems WD1_5 and WD2_5 perform better than the implementa-

tions with higher parameter values. However, for the OR Library problems with $m = 10$, the convergence of the solver drops significantly, and for these test problems, the implementations with higher parameter values such as WD1_15, WD2_15, or HYB_20_15 perform better in this case.

When we compare different implementations, the hybrid implementation performs better than other implementations in most cases. This conclusion is expected since the hybrid implementation is designed to overcome the shortcomings of the first two implementations.

Chapter 5

Multi-resource Combinatorial Auction Problems

5.0.4 Problem Description

One application of the Multi-dimensional Knapsack Problem is the *Winner Determination Problem* (WDP) of the general combinatorial auctions (CA). In general combinatorial auctions, bidders bids on bundles of goods [35]. The objective of this problem is maximizing the revenue by determining which bidders to accept given the limited number of goods in the auction. This type of auction became popular when it was used as a winner determination mechanism for FCC spectrum auctions and airline time slot auctions while letting airlines bid simultaneously [50, 61]. There are other applications of this type of auction [61]. In this study we focus on determining the winners of a multi-unit CA. This problem is a generic version of the single-unit CA. Each bidder can bid on bundles of unrestricted number of goods. There are identical copies of the same good in this type of auction [36]. Optimal design of these auctions became a widely studied topic after Internet auctions became popular. We extend the example given in [36] by making it more general. In electronics manufacturing auctioning, a manufacturer might want to sell 1000 TVs, 500 VCRs, 500 freezers, and 600 washers. Each bidder is allowed to bid on any number of any goods. One bidder might bid on 200 TVs and 120 VCRs, while another bidder might bid on 200 freezers, 300 washers and 10 TVs. The optimization problem is determining which bids to accept in order to maximize the total revenue of the auction holder. The decision is whether to accept a bid or not. The number of each good to be auctioned is given. A bundle of a bidder is determined by the price of the bundle and number of each good that is included in the bundle.

5.0.5 Solution Methodologies

WDP is an *NP*-hard problem. Some of the algorithms to solve these problems have better than average performance than average; however, algorithm performance is mostly affected by the problem instance. Leyton-Brown et al. [34] give a good description on how to evaluate different ways to solve this problem. The main idea in this article is to train the algorithm choosing strategies based on some randomly but logically generated test suites. According to this study, CPLEX gives far better results than specialized algorithms. This result is not very surprising since WDP is a very generic problem and commercial solvers have many components built in that can address different ill-conditions. In the next subsection, we propose a new test suite generator for the WDP problem. We use this test suite to measure the performance changes of the SAS/OR MILP solver with the proposed implementations.

5.1 Specialized CAT Suite

In this section we describe the test suite that is used to evaluate the performance of the canonical cuts within the SAS/OR MILP solver. For comparing the algorithm performance, there was not a universal benchmark to compare the results until Leyton-Brown et al. published their work on building a test generator called CATS [35]. This test generator is open to the public and includes different distributions which generate test problems for different applications of combinatorial auctions. However, none of these distributions is addressing the type of WDP problem we study. We propose a new test generation routine that adapts some ideas of the test generator proposed by Freville and Plateu [21] and CATS to generate realistic problems for our study. The nature of WDP for multi-unit combinatorial auctions imposes correlation between constraint and objective coefficient. As mentioned earlier, this type of problem is in the difficult problem category.

5.1.1 Test Suite Description

The inputs for the test generator are as follows:

- The number of goods in the auction
- The number of bidders joining the auction
- The number of categories for goods
- The upper bound for perceived value variability range per bid
- Available resource adjustment coefficient

The first things the test generator determines are which goods belong to each category and the relative value coefficient per category. The category a good belongs to is determined randomly. While determining the value of a category, we assume that goods in a category are in the same value range and it is possible to represent the value of every good in a category with one coefficient. The value coefficient of a category, v_{cat}, is a relative measure of the price of the corresponding category to other categories. We generate one random number per category and normalize this number. After fixing the category information, bundles per bidder are determined. There are multiple things to determine to generate the bundle information of a bidder. The first thing is the set of categories a bidder focuses on. In real life, bidders usually have a focus on a certain group of products. As an example, in an electronics manufacturing auction, a bidder might represent a store and might focus on home appliances more than PC related goods. For each bidder, we randomly determine which categories to focus on. The distribution to generate the demand for focused categories is required to always generate larger

values than the distribution to generate the demand for goods that belong to other groups. For example, the store joining the auction which mainly focuses on home appliances might still be interested in bidding on a small number of PC related goods, since it is a common practice for stores to carry items from categories they do not focus on as well. The value information of a bundle is determined by multiplying the number of items in a category by category value coefficient and summing all these values with a randomly generated number bounded by the upper bound for perceived value variability range. The final thing to determine for an auction is the number of available resources. If the number of items per good is too high, then the optimal solution might get close to accepting all the bids. We want to avoid trivial auction settings by multiplying the total number of items requested for the corresponding good by the available resource adjustment coefficient. It is possible to make a problem very difficult by decreasing the upper bound on the perceived value variability coefficients or by increasing the resource available to a certain extent to increase the number of feasible combinations of bids.

5.2 Evaluating the Proposed Canonical Cut Implmentations over Combinatorial Auction Problems

In this section we evaluate the performance of different implementations of the depth-1 and depth-2 canonical cuts over the specialized CAT suite which is explained in Section 5.1. We compare the performance of three implementations of integrating depth-1 and depth-2 canonical cuts into SAS/OR MILP solver as explained in Section 4.3. In Section 4.3, we conclude on the best parameter based on the experimental results over OR Library problems. The size of the problems we generate in this section have size similar to some of the OR Library problems. Therefore, we use these parameter tunings for evaluating the performance changes over these problems. We also impose a 30-minute time limit on the solver for the experimental runs in this section. The first subsection explains the parameter setting for the test generation. The rest of the subsections focus on performance evaluation. In the second subsection, we compare the solution status statistics. This gives the frequency of success per experiment. In the third subsection, we analyze the performance profile curve for each experiment. The last subsection focuses on the relative gap for non-convergent cases. In other words, we evaluate the performance of the solution if the solver versions are not successful.

5.2.1 Parameter Description for Test Suite Generation

We generate test problems using the procedure explained in Section 5.1. Since this procedure has some similarities to the test generator described in [20, 21], our parameter setting is influenced by these works. We use a Uniform distribution for random number generation. Table 5.1 shows the type of distributions used in the problem generation. There are 500 bids and

Table 5.1: Random number distributions for test generation

Name	Distribution
Value coefficient per category	$U[0, 1]$
Perceived value variability per bidder	$U[0, 500]$
Demand for focus category per bidder	$U[500, 1000]$
Demand for other category per bidder	$U[100, 200]$

30 different types of goods. The resource adjustment coefficient is 0.25. The number of goods categories is 3 for 19 problems, 4 for 19 problems, and 5 for 19 problems.

5.2.2 Success Frequency

Table 5.2 summarizes the solution status of the experimental runs over the specialized CA test suite. For this test suite, the solver either reports optimal within relative gap ($1e^{-6}$) or it reaches the time limit. The column header for the former is OPTIMAL_RGAP, for the latter it is TIME_LIMIT. The first row gives the statistics for the solver before any implementation is integrated. For all the implementations, results are better than WO. The best result is from the pure depth-2 implementation with turn off parameter of 20. This implementation is able to solve 10 more problems than WO. Looking at these statistics, WD2_20, WD2_30, and HYB_30_25 give the best outcome.

Table 5.2: Solution status summary for specialized CA test suite

Implementation	OPTIMAL_RGAP	TIME_LIMIT
WO	21	36
WD1_25	25	32
WD1_30	25	32
WD2_20	31	26
HYB_25_20	29	28

5.2.3 Performance Profiles for Time to Converge and Relative Gap

Figure 5.1 shows the performance profile curves for time to converge. All of the implementations perform better than WO. The curves that belong to the implementations with only depth-1 canonical cuts are always under the curves that belong to other implementations, so these are not competitive with the other implementations. The best performing implementation is WD2_20 and the second is HYB_25_20.

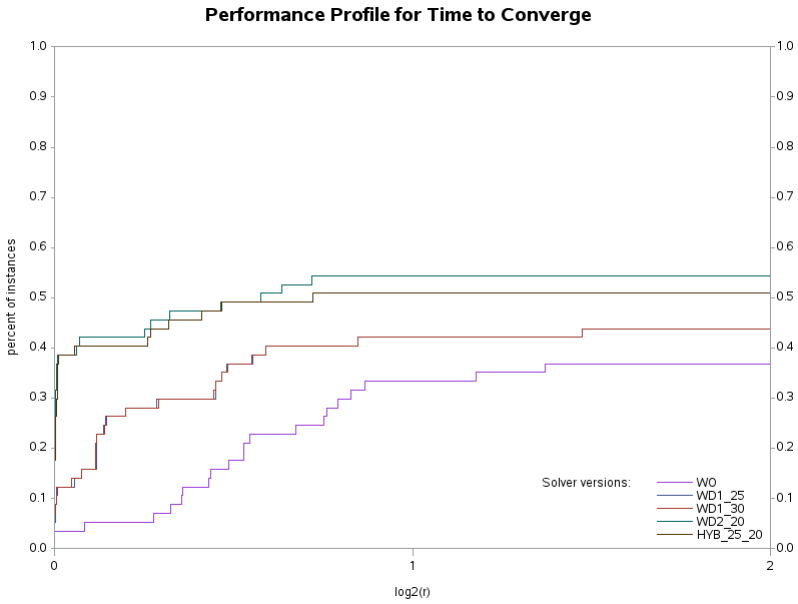


Figure 5.1: Performance profile for time to converge for specialized CATS

Figure 5.2.3 shows the performance profile curves for the relative gap among the test problems that do not converge within the given time. This analysis gives an idea about the quality of the solution even though the solver might not converge. All of the implementations perform better than the version without canonical cuts. The best performing implementations are WD2_20 and HYB_25_20. The performance of WD1_25 is the same as WD1_30. These results are consistent with the performance profiles of time to converge.

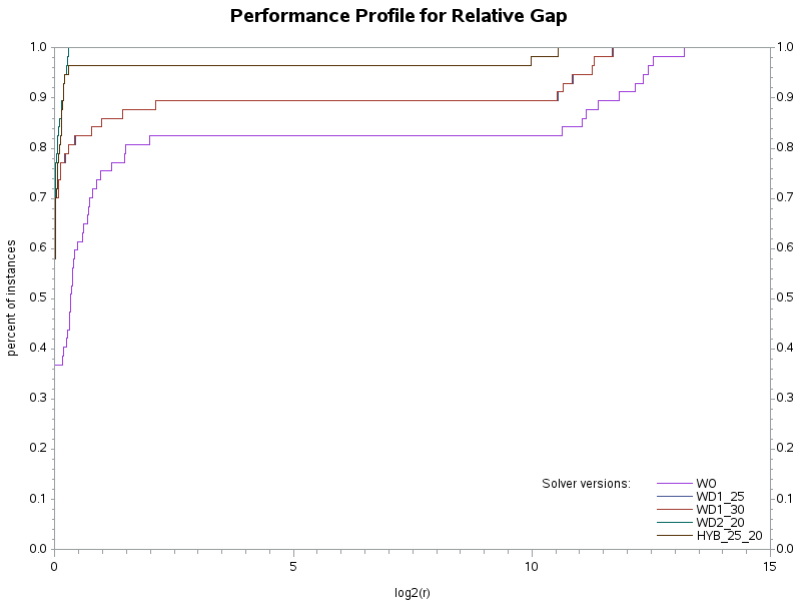


Figure 5.2: Performance profile for relative gap for specialized CATS

5.3 Conclusion

We use the best performing solver versions for testing the effectiveness of the canonical cuts on WDP. Overall, it is shown that canonical cuts are very effective solving WDP problems generated by the proposed test generator which is explained in subsection 5.1. There is always performance increase over the solver without any canonical cuts. The test results in this section differ from the results of the experiments which are done over general MKP problems. We conclude that hybrid implementation works better over the general MKP problems, although over combinatorial auction problems, implementation with depth 2 canonical cuts performs slightly better than hybrid approaches. One consistent results is that lowest performing implementations are the ones with depth-1 canonical cuts. It is worth mentioning that performance evaluations are consistent over different performance measures.

Chapter 6

Conclusions and Future Work

The general trend in integer programming solving methodologies is going towards customizing the solving methodologies for special structure problems. In light of this trend, we started this study by aiming to solve the Multi-dimensional Knapsack Problem effectively.

Our approach was using some geometric insights to design an efficient algorithm for solving MKP. Due to their geometric properties, we integrated canonical cuts into our solver. We extended the theory on canonical cuts. This extension improved the effectiveness of the cut. We introduced the concept of depth-k canonical cuts and proposed various implementations of these cuts. Proposed implementations were shown to have the potential to improve both primal and dual bounds. With these implementations, we introduced the parametric control of the separation problem complexity limit during cut generation, in addition to hybrid usage of various depth level canonical cuts. These implementations showed how to use these cuts practically and efficiently.

We integrated these cuts into the SAS/OR MILP solver. This integration improved the performance of the SAS/OR MILP solver on general MKP and WDP for multi-resource combinatorial auction problems.

There are three main areas in which we would like to continue our work. The first is the efficiency improvements of the current implementations. For this part, we will focus on cut generation improvements such as using a more sophisticated solver instead of implicit enumeration and cut quality improvements such as incorporating conflict analysis. The reason we will focus on this area is to be able to generate better cuts using the same effort. The second area is doing an extensive analysis to understand the changes in the effectiveness of the proposed implementations on different types of problems. In this study, we did this type of analysis; however, we believe deeper analysis will increase the practicality of these implementations.

Lastly, we would like to do more experimentation using the MIPLIB library. In this extension, canonical cuts will be added over the proper structure of the test problems. This extension

will give us a more global comparison of the performance.

REFERENCES

[1] Tobias Achterberg. *Constraint integer programming.* PhD thesis, Technische Universitat Berlin, 2007.

[2] Yalcin Akcay, Haijun Li, and Susan Xu. Greedy algorithm for the general multidimensional knapsack problem. *Annals of Operations Research*, 150:17–29, 2007.

[3] David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William Cook. *The traveling salesman problem, a computational study.* Princeton Series in APPLIED MATHEMATICS, 2006.

[4] E. Balas, S. Ceria, G. Cornujols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19(1):1–9, 1996.

[5] Egon Balas, Fred Glover, and Stanley Zionts. An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, 13(4):517–549, 1965.

[6] Egon Balas and Robert Jeroslow. Canonical cuts on the unit hypercube. *SIAM Journal on Applied Mathematics*, 23:61–69, 1978.

[7] Egon Balas and Eitan Zemel. An algorithm for large zero-one knapsack problems. *Operations Research*, 28(5):1130–1154, 1980.

[8] Paulo Barcia. The bound improving sequence algorithm. *Operations Research Letters*, 4(1):27–30, 1985.

[9] Paulo Barcia and Soren Holm. A revised bound improvement sequence algorithm. *European Journal of Operational Research*, 36(2):202–206, 1988.

[10] Dimitris Bertsimas and Ramazan Demir. An approximate dynamic programming approach to multidimensional knapsack problems. *Management Science*, 48(4):550–565, 2002.

[11] P.C. Chu and J.E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.

[12] Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102:71–90, 2005.

[13] George Dantzig. *Linear programming and extensions.* Princeton University Press, August 1998.

[14] Elizabeth D. Dolan and Jorge J. More. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

[15] Hugh Everett. Generalized lagrange multiplier method for solving problems of optimum allocation of resources. *Operations Research*, 11(3):399–417, 1963.

[16] Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.

[17] Krzysztof Fleszar and Khalil S. Hindi. Fast, effective heuristics for the 0-1 multi-dimensional knapsack problem. *Comput. Oper. Res.*, 36:1602–1607, May 2009.

[18] Uwe H. Suhl Franz Wesselmann. Implementation techniques for cutting plane management and selection. *Optimization Methods and Software*, submitted, 2008.

[19] Arnaud Freville. The multidimensional 0-1 knapsack problem: an overview. *European Journal of Operational Research*, 155(1):1–21, 2004.

[20] Arnaud Freville and Gerard Plateau. The 0-1 bidimensional knapsack problem: toward an efficient high-level primitive tool. *Journal of Heuristics*, 2:147–167, 1996.

[21] Arnaud Freville and Grard Plateau. An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem. *Discrete Applied Mathematics*, 49(1-3):189–212, 1994.

[22] V. Gabrel, A. Knippel, and M. Minoux. Exact solution of multicommodity network optimization problems with general step cost functions. *Operations Research Letters*, 25(1):15–23, 1999.

[23] B. Gavish and H. Pirkul. Allocation of databases and processors in a distributed computing system. *Management of Distributed Data Processing*, pages 215–231, 1982.

[24] Bezalel Gavish and Hasan Pirkul. Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Mathematical Programming*, 31:78–105, 1985.

[25] P. C. Gilmore and R. E. Gomory. The theory and computation of knapsack functions. *Operations Research*, 14(6):1045–1074, 1966.

[26] Fred Glover. A multiphase-dual algorithm for the zero-one integer programming problem. *Operations Research*, 13(6):879–919, 1965.

[27] R. E. Gomory. An algorithm for mixed integer problem. Technical report, Technical Report RM-2597, The Rand Corporation, 1960.

[28] R.E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[29] Harvey J. Greenberg and William P. Pierskalla. Surrogate mathematical programming. *Operations Research*, 18(5):924–939, 1970.

[30] Sad Hanafi and Fred Glover. Exploiting nested inequalities and surrogate constraints. *European Journal of Operational Research*, 179(1):50–63, 2007.

[31] Seymour Kaplan. Solution of the lorie-savage and similar integer programming problems by the generalized lagrange multiplier method. *Operations Research*, 14(6):1130–1136, 1966.

[32] Fatma Kilinc Karzan, George Nemhauser, and Martin Savelsbergh. Information-based branching schemes for binary linear mixed integer problems. *Mathematical Programming Computation*, 1:249–293, 2009.

[33] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[34] Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim McFadden, and Yoav Shoham. Boosting as a metaphor for algorithm design. In *Principles and Practice of Constraint Programming*, volume 2833 of *Lecture Notes in Computer Science*, pages 899–903. Springer Berlin / Heidelberg, 2003.

[35] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *In ACM Conference on Electronic Commerce*, pages 66–76, 2000.

[36] Kevin Leyton-Brown, Yoav Shoham, and Moshe Tennenholtz. An algorithm for multi-unit combinatorial auctions. In *In Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 56–61, 2000.

[37] James H. Lorie and Leonard J. Savage. Three problems in rationing capital. *The Journal of Business*, 28(4):229–239, 1955.

[38] Richard Loulou and Eleftherios Michaelides. New greedy-like heuristics for the multidimensional 0-1 knapsack problem. *Operations Research*, 27(6):1101–1114, 1979.

[39] M. J. Magazine and Osman Oguz. A heuristic algorithm for the multidimensional zero-one knapsack problem. *European Journal of Operational Research*, 16(3):319–326, 1984.

[40] Roy E. Marsten and Thomas L. Morin. A hybrid approach to discrete mathematical programming. *Mathematical Programming*, 14:21–40, 1978.

[41] Silvano Martello and Paolo Toth. A new algorithm for the 0-1 knapsack problem. *Management Science*, 34(5):633–644, 1988.

[42] Alexander Martin. General mixed integer programming: computational issues for branch-and-cut algorithms. *Computational Combinatorial Optimization*, 2241:1–25, 2001.

[43] G. L. Nemhauser and Z. Ullmann. Discrete dynamic programming and capital allocation. *Management Science*, 15(9):494–505, 1969.

[44] M. W. Padberg, T. J. van Roy, and L. A. Wolsey. Valid linear inequalities for fixed charge problems. *Operations Research*, 33(4):842–861, 1985.

[45] David Pisinger. An expanding-core algorithm for the exact 0-1 knapsack problem. *European Journal of Operational Research*, 87(1):175–187, 1995.

[46] Jakob Puchinger, Gunther Raidl, and Ulrich Pferschy. The core concept for the multidimensional knapsack problem. *Evolutionary Computation in Combinatorial Optimization*, 3906:195–208, 2006.

[47] Jakob Puchinger, Günther R. Raidl, and Gruber Martin. Cooperating memetic and branch-and-cut algorithms for solving the multidimensional knapsack problem. *Proceedings of MIC 2005, the 6th Metaheuristics International Conference*, 2005.

[48] Jakob Puchinger, Günther R. Raidl, and Ulrich Pferschy. The multidimensional knapsack problem: structure and algorithms. *INFORMS Journal on Computing*, 22:250–265, April 2010.

[49] Gunther Raidl and Jens Gottlieb. Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary Computation Journal*, 13:441–475, 2005.

[50] S. J. Rassenti, V. L. Smith, and R. L. Bulfin. A combinatorial auction mechanism for airposrt time slot allocation. *Bell Journal of Economics*, 13:402–417, 1982.

[51] SAS. *SAS/OR 9.3 User's Guide: Mathematical Programming*. SAS Ins., 2011.

[52] Shizuo Senju and Yoshiaki Toyoda. An approach to linear programming with 0-1 variables. *Management Science*, 15(4):B196–B207, 1968.

[53] Wei Shih. A branch and bound method for the multiconstraint zero-one knapsack problem. *Journal of Operations Research Society*, 30:369–378, 1979.

[54] João P. Marques Silva and Karem A. Sakallah. Grasp a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, ICCAD '96, pages 220–227, Washington, DC, USA, 1996. IEEE Computer Society.

[55] A. L. Soyster, B. Lev, and W. Slivka. Zero-one programming with many variables and few constraints. *European Journal of Operational Research*, 2(3):195–201, 1978.

[56] Arne Thesen. A recursive branch and bound algorithm for the multidimensional knapsack problem. *Naval Research Logistics Quarterly*, 22(2):341–353, 1975.

[57] Michel Vasquez and Jin-Kao Hao. A hybrid approach for the 01 multidimensional knapsack problem. *In Proceedings of the International Joint Conference on Artificial Intelligence*, 2001.

[58] Michel Vasquez and Yannick Vimont. Improved results on the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 165(1):70–81, 2005.

[59] A. Volgenant and J. A. Zoon. An improved heuristic for multidimensional 0-1 knapsack problems. *The Journal of the Operational Research Society*, 41(10):963–970, 1990.

[60] A. Volgenant and I. Y. Zwiers. Partial enumeration in heuristics for some combinatorial optimization problems. *The Journal of the Operational Research Society*, 58(1):73–79, 2007.

[61] Sven de Vries and Rakesh Vohra. Combinatorial auctions: a survey. Discussion Papers 1296, Northwestern University, Center for Mathematical Studies in Economics and Management Science, 2000.

[62] H. Martin Weingartner and David N. Ness. Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research*, 15(1):83–103, 1967.

[63] F.P. Wyman. Binary programming: a decision rule for selecting optimal vs heuristic techniques. *Computer Journal*, 16:135–140, 1973.

APPENDICES

Appendix A

Performance Measure Distributions

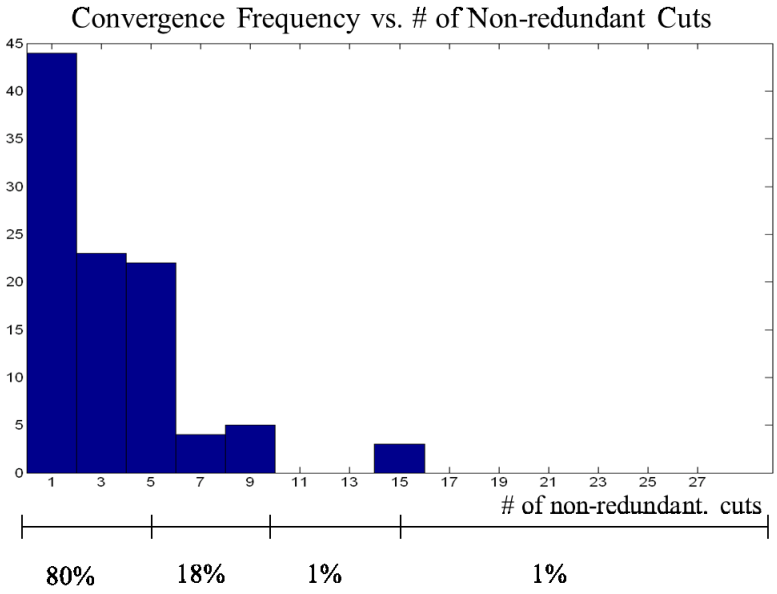
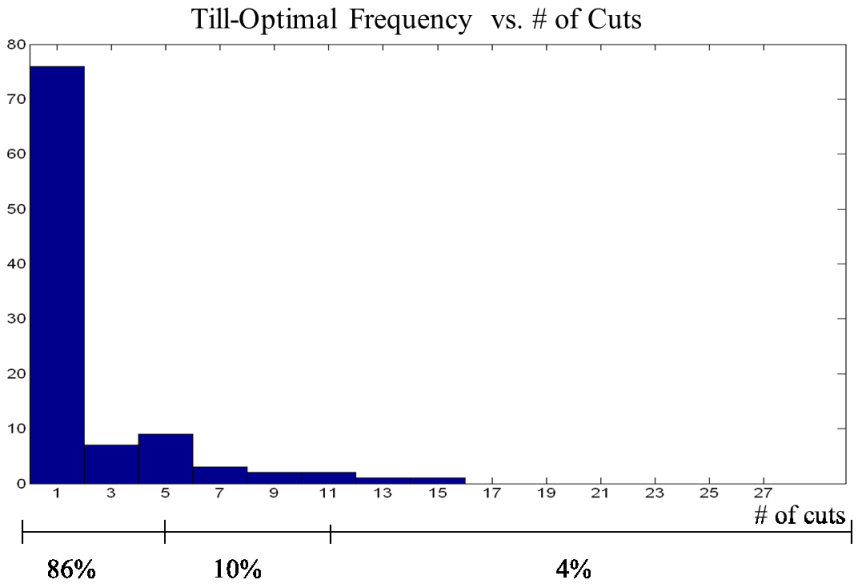
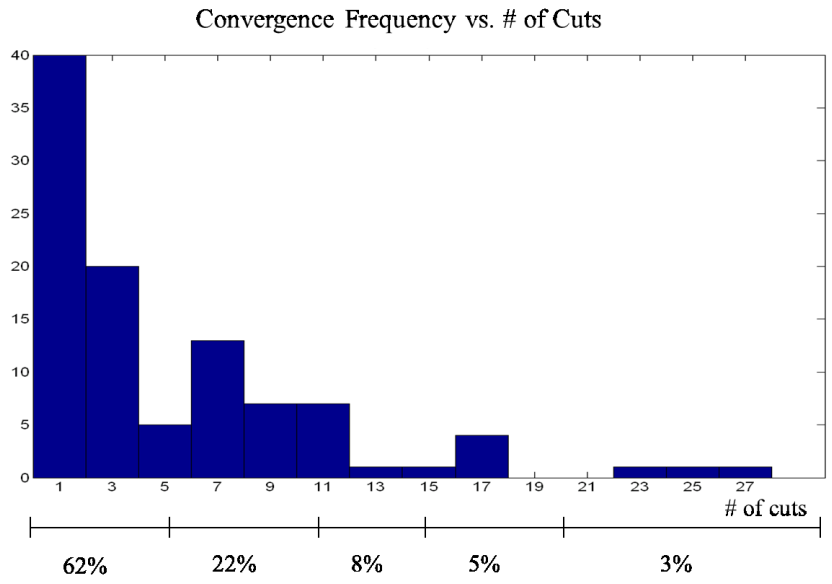


Figure A.1: Number of nonredundant cuts at termination

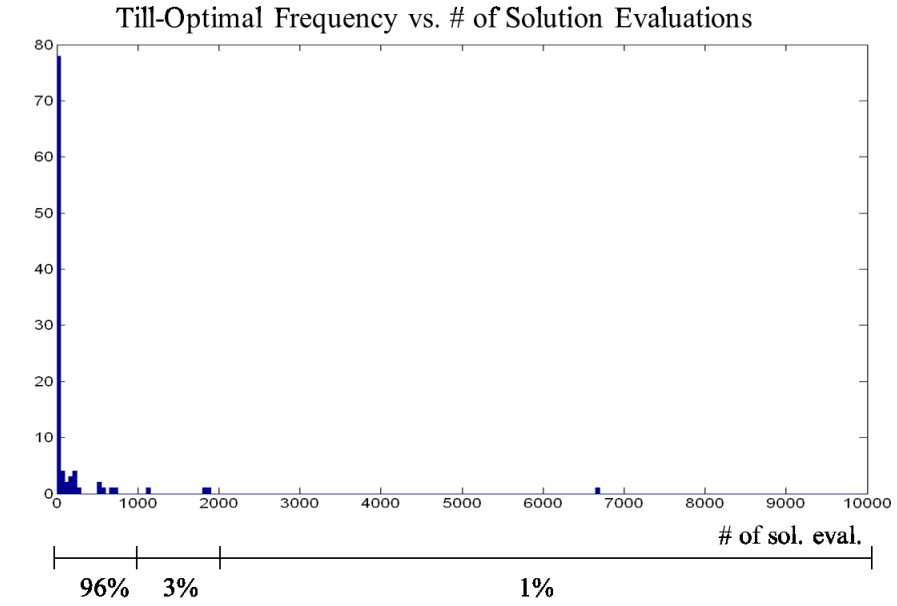


(a) Until optimal solution found

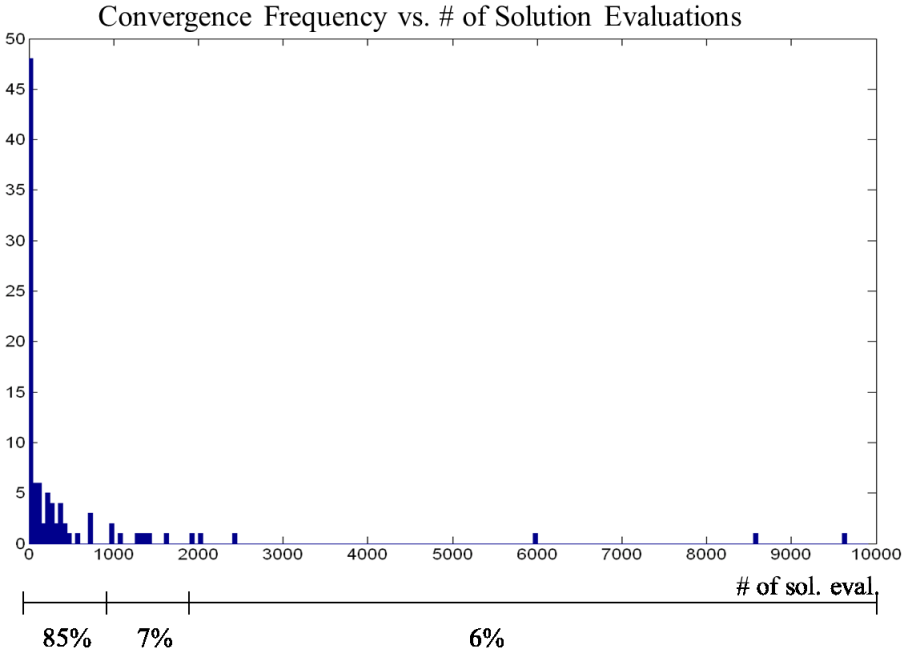


(b) Until convergence

Figure A.2: Number of cuts added

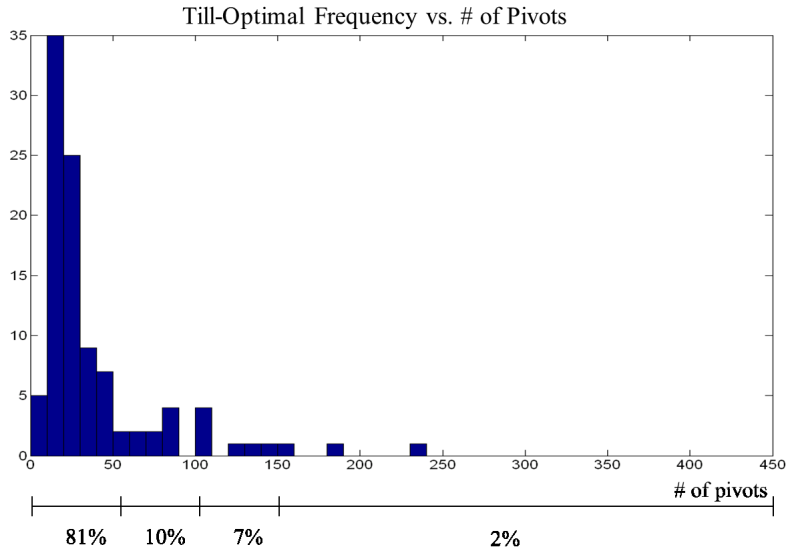


(a) Until optimal solution found

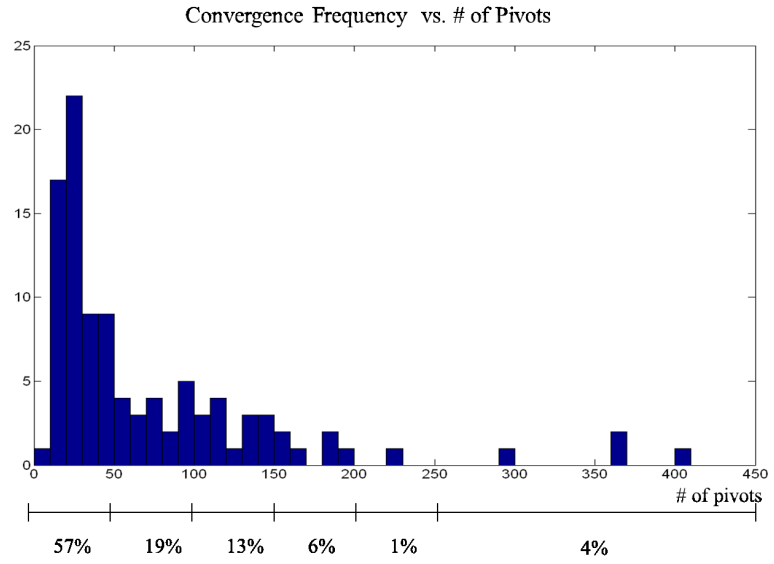


(b) Until convergence

Figure A.3: Number of solutions evaluated during separation problem solve



(a) Until optimal solution found

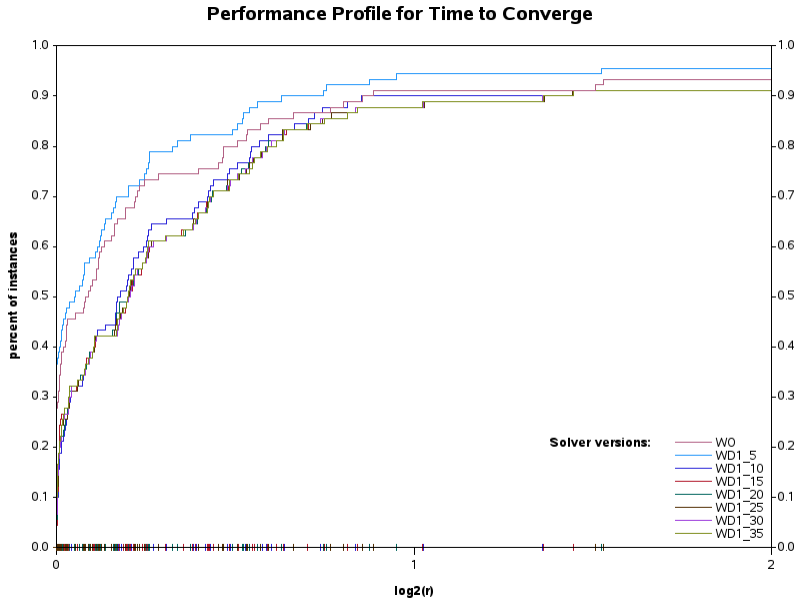


(b) Until convergence

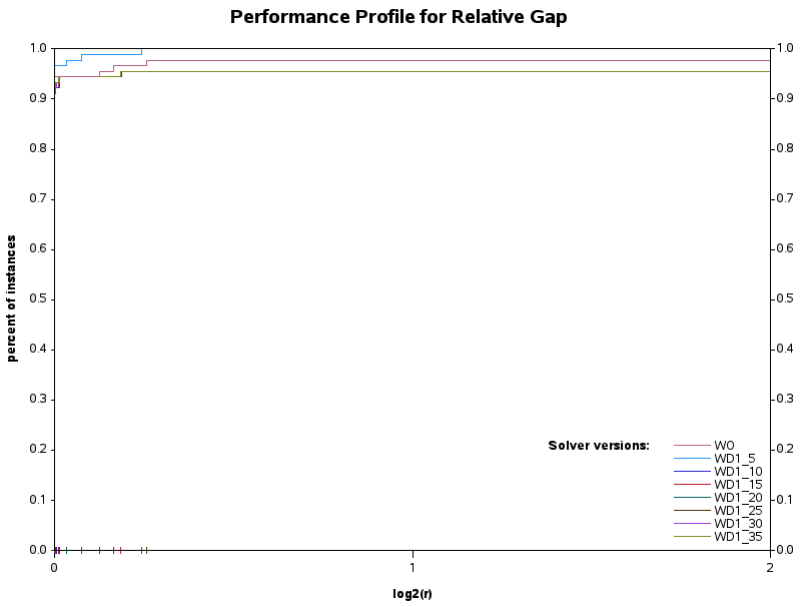
Figure A.4: Number of solutions evaluated during separation problem solve

Appendix B

Performance Profiles of the SAS/OR Solver with the Proposed Implementations Over OR Library Problems

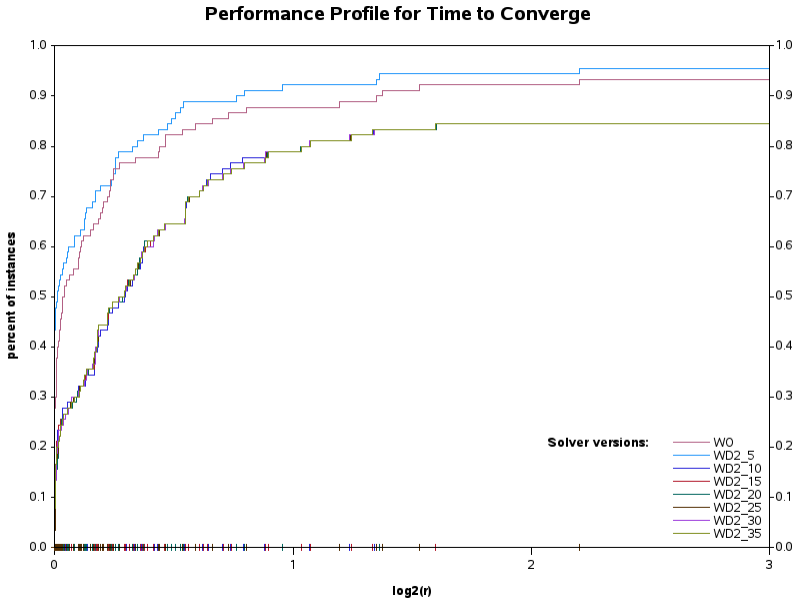


(a) Time to converge

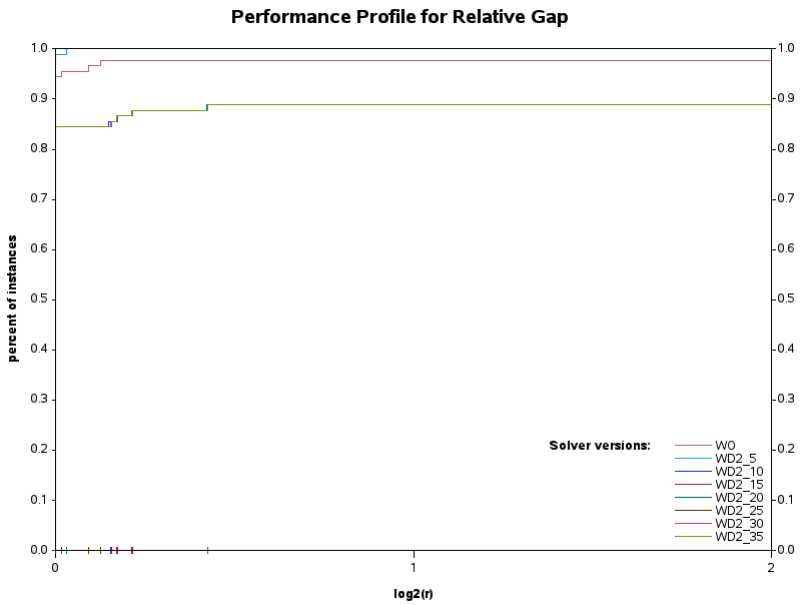


(b) Relative gap

Figure B.1: Performance profiles for ORLIB test problems with $m = 5$ of implementation 1



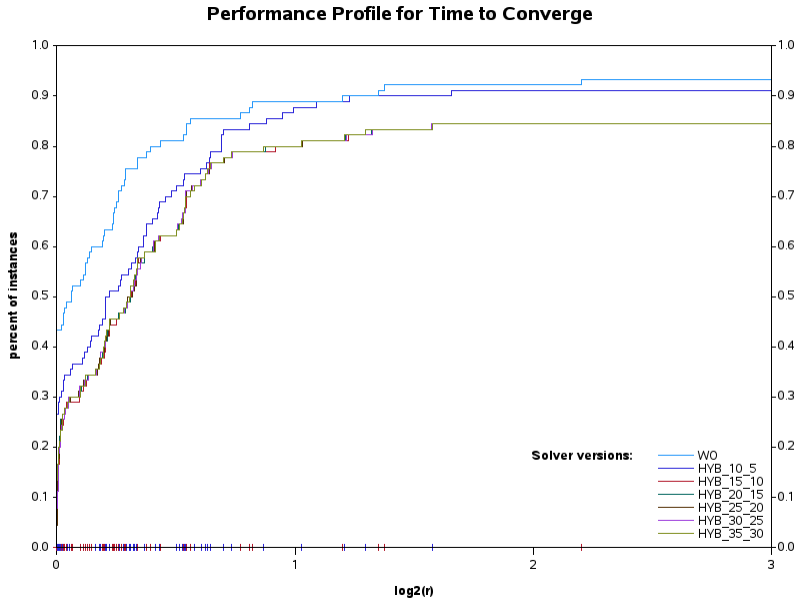
(a) Time to converge



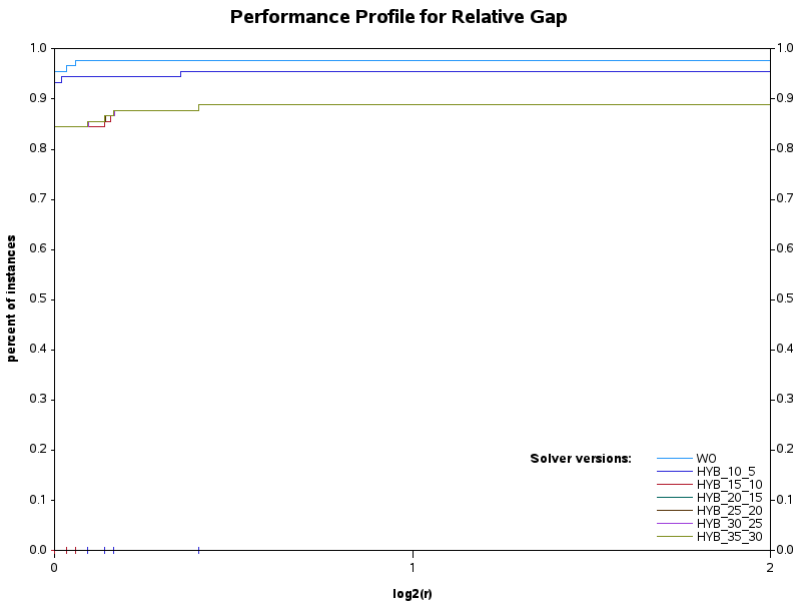
vvc va

(b) Relative gap

Figure B.2: Performance profiles for ORLIB test problems with $m = 5$ of implementation 2

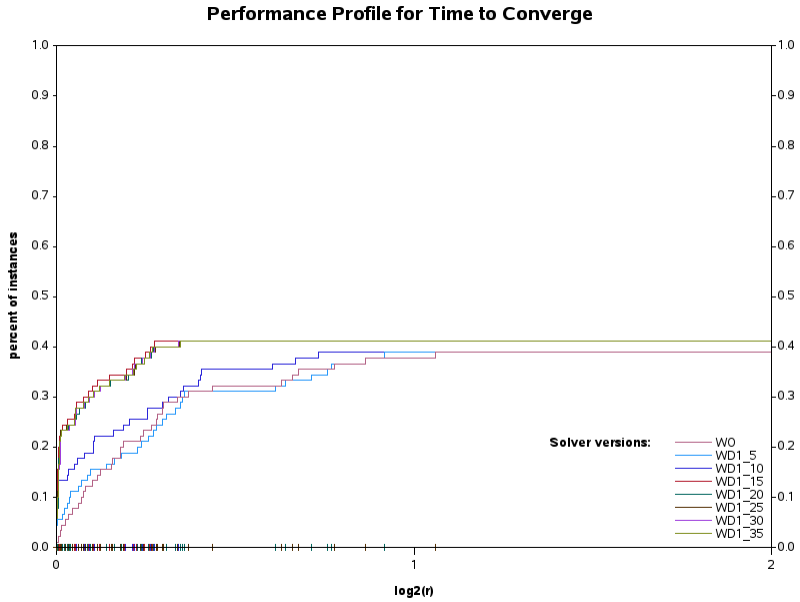


(a) Time to converge

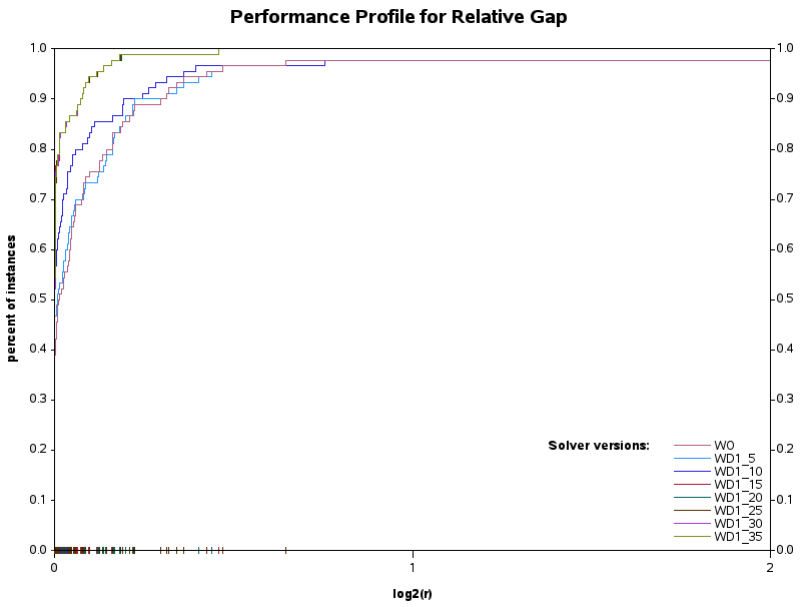


(b) Relative gap

Figure B.3: Performance profiles for ORLIB test problems with $m = 5$ of implementation 3

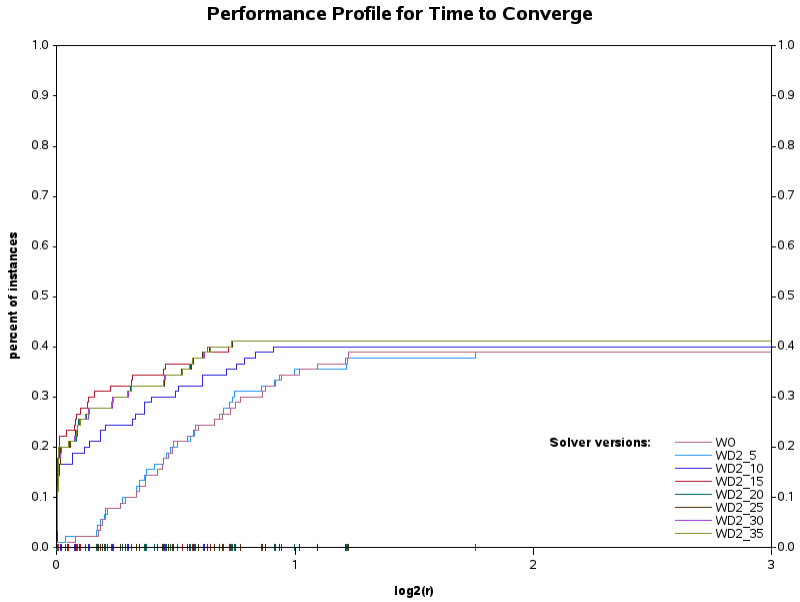


(a) Time to converge

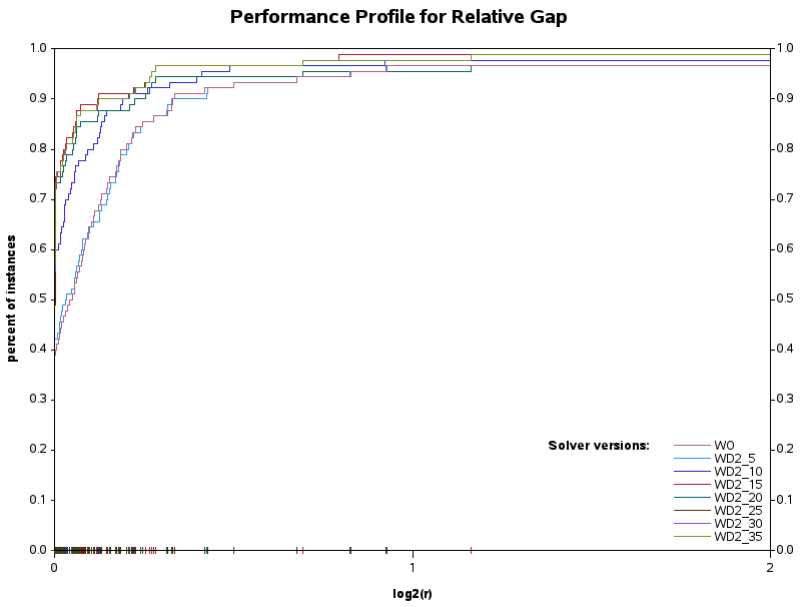


(b) Relative gap

Figure B.4: Performance profiles for ORLIB test problems with $m = 10$ of implementation 1

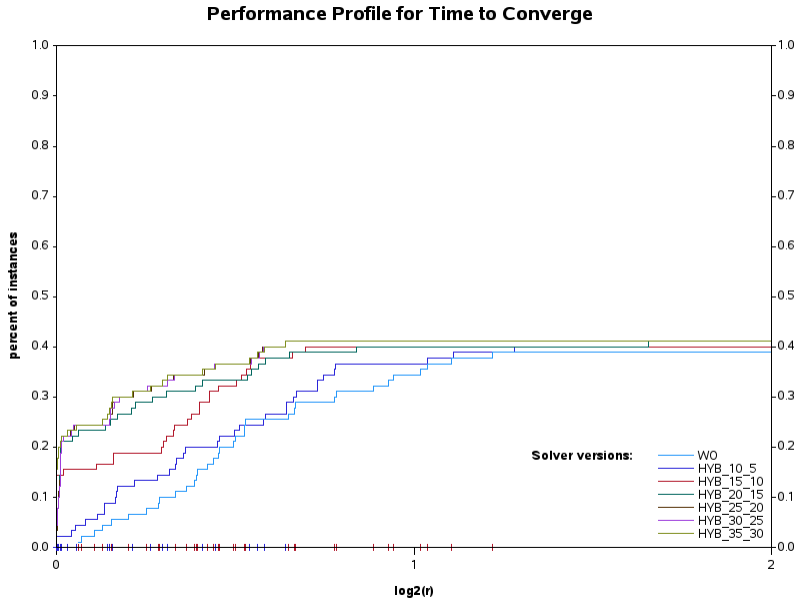


(a) Time to converge

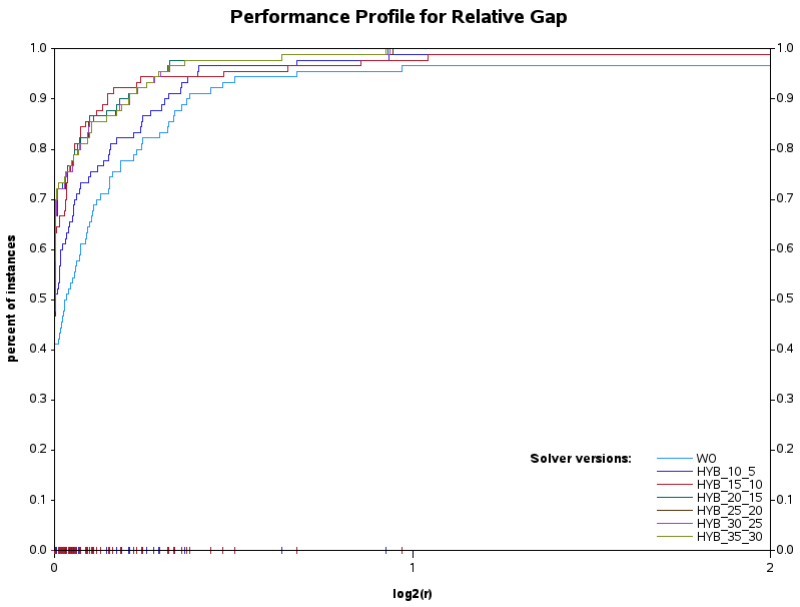


(b) Relative gap

Figure B.5: Performance profiles for ORLIB test problems with $m = 10$ of implementation 2

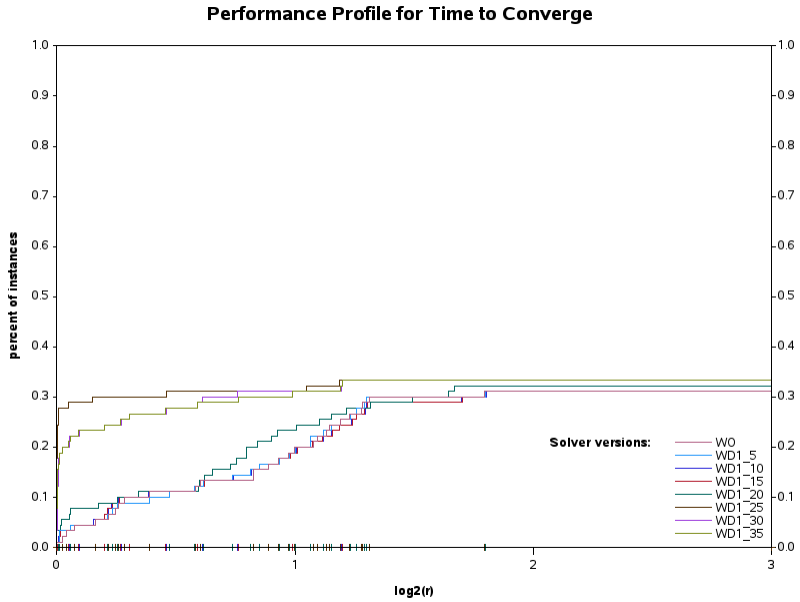


(a) Time to converge

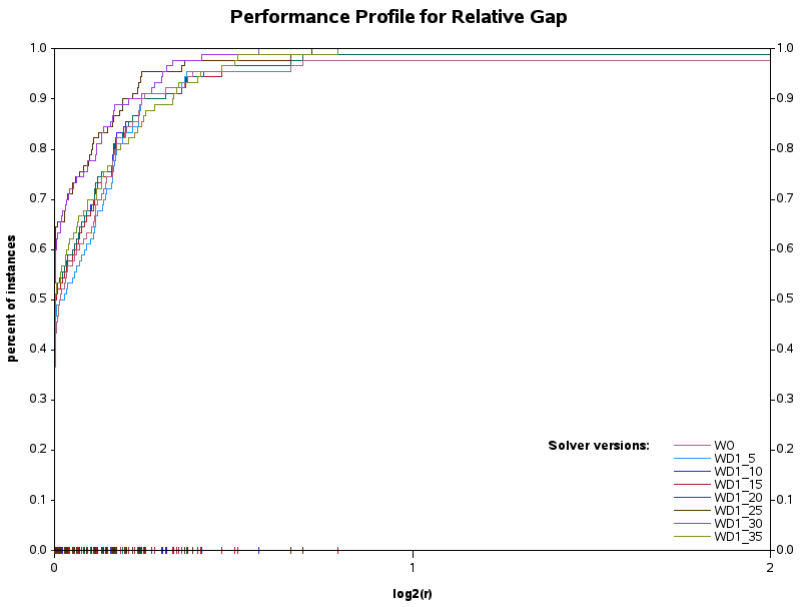


(b) Relative gap

Figure B.6: Performance profiles for ORLIB test problems with $m = 10$ of implementation 3

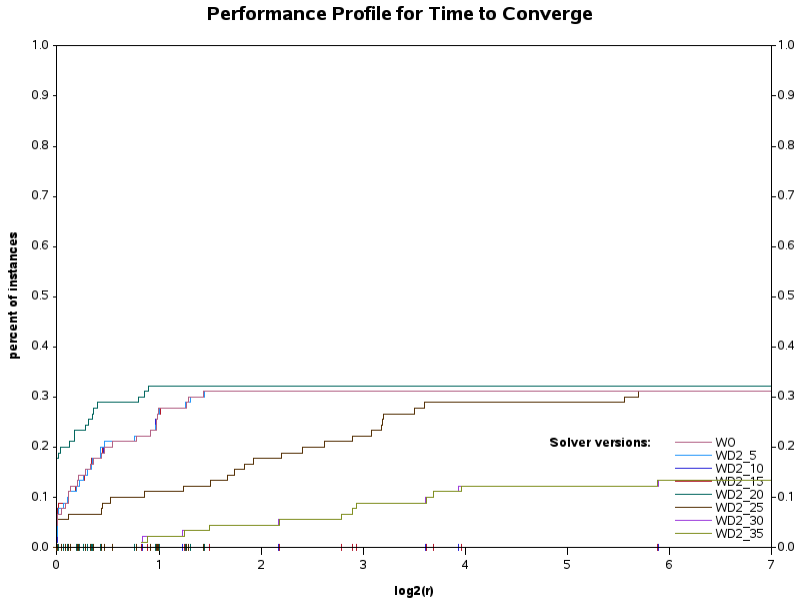


(a) Time to converge

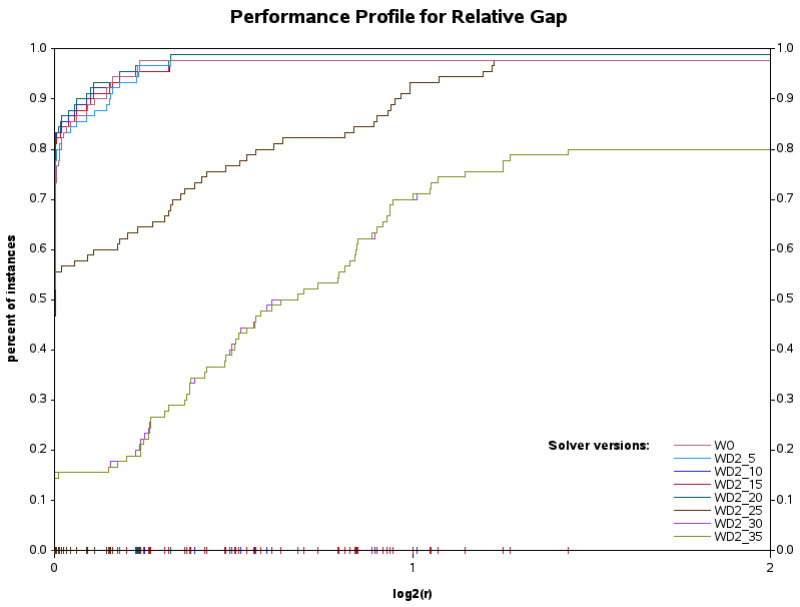


(b) Relative gap

Figure B.7: Performance profiles for ORLIB test problems with $m = 30$ of implementation 1

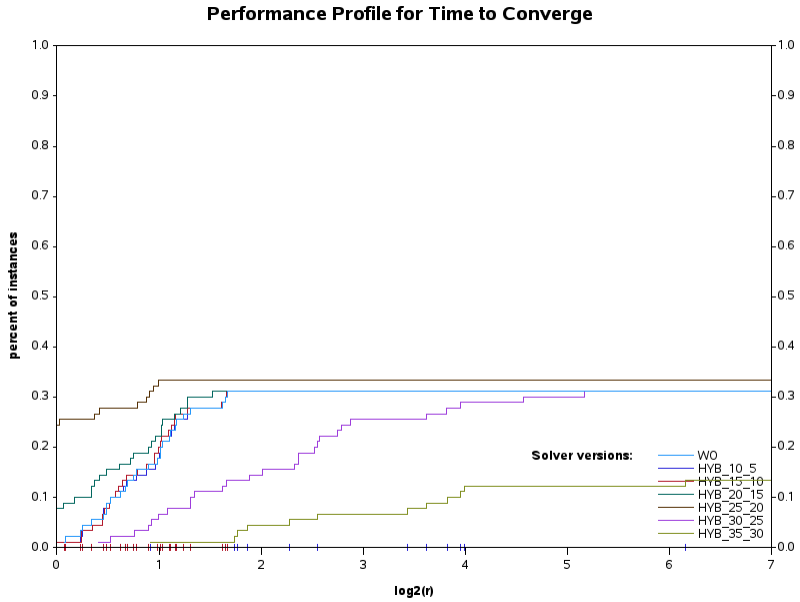


(a) Time to converge

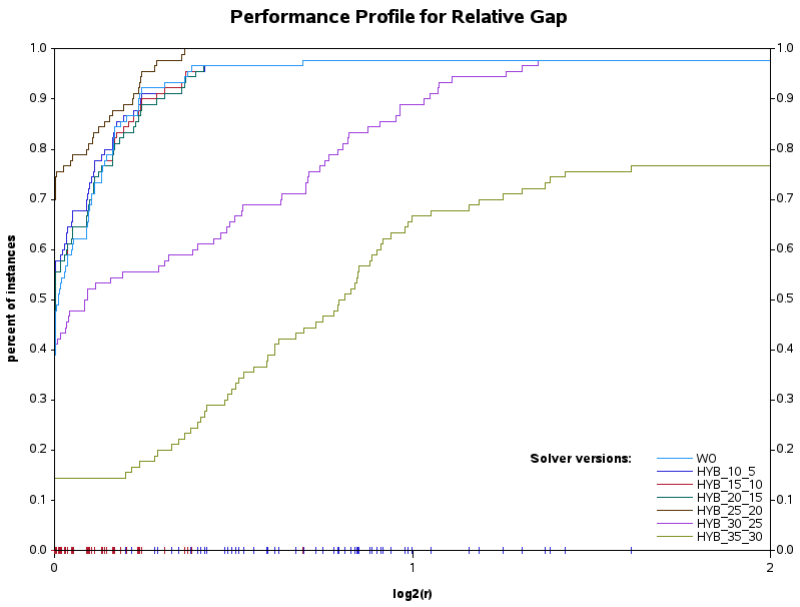


(b) Relative gap

Figure B.8: Performance profiles for ORLIB test problems with $m = 30$ of implementation 2



(a) Time to converge



(b) Relative gap

Figure B.9: Performance profiles for ORLIB test problems with $m = 30$ of implementation 3

Appendix C

Solution Improvements of OR Library Problems with the Best Implementation

This chapter shows the improvements of the solution found by the solver with the best implementations. In literature, first Chu and Beasley [11] reported results of their proposed genetic algorithm for solving these problems. In the tables we use GA acronym to refer to these results. Second most significant results are reported by Vasquez and Hao [57]. They proposed tabu search for solving these problems. In the tables we use TS to refer to these results. In 2005, Vasquez and Vimont [58] enhanced [57] by using some fixation. The results of this work is referred to as Fix+TS. We use SAS/WO to refer to SAS/OR MILP solver without the proposed the implementations and SAS/W for the version with the best implementation.

Table C.1: Best known solutions found for $n = 500$ and $m = 5$

No	GA[11]	TS[57]	Fix+TS[58]	SAS/WO	SAS/W
1	120130	120134	120148	120148	120148
2	117837	117864	117879	117879	117879
3	121109	121112	121131	121131	121131
4	120798	120804	120804	120804	120804
5	122319	122319	122319	122319	122319
6	122007	122024	122011	122024	122024
7	119113	119127	119127	119127	119127
8	120568	120568	120568	120568	120568
9	121575	121575	121575	121586	121586
10	120699	120707	120717	120717	120717
11	218422	218428	218426	218428	218428
12	221191	221191	221202	221202	221202
13	217534	217534	217542	217542	217542
14	223558	223558	223560	223560	223560
15	218962	218966	218965	218966	218966
16	220514	220530	220527	220530	220530
17	219987	219989	219989	219989	219989
18	218194	218194	218215	218215	218215
19	216976	216976	216976	216976	216976
20	219693	219704	219719	219719	219719
21	295828	295828	295828	295828	295828
22	308077	308083	308086	308086	308086
23	299796	299796	299796	299796	299796
24	306476	306478	306480	306480	306480
25	300342	300342	300342	300342	300342
26	302560	302561	302571	302571	302571
27	301322	301329	301339	301339	301339
28	306430	306454	306454	306454	306454
29	302814	302822	302828	302828	302828
30	299904	299904	299910	299904	299904

Table C.2: Best known solutions found for $n = 500$ and $m = 10$

No	GA[11]	TS[57]	Fix+TS[58]	SAS/WO	SAS/W
1	117726	117779	117811	117766	117790
2	119139	119190	119232	119182	119206
3	119159	119194	119215	119211	119211
4	118802	118813	118813	118813	118825
5	116434	116462	116509	116471	116509
6	119454	119504	119504	119475	119475
7	119749	119782	119827	119777	119797
8	118288	118307	118329	118270	118323
9	117779	117781	117815	117781	117781
10	119125	119186	119231	119191	119212
11	217318	217343	217377	217318	217377
12	219022	219036	219077	219038	219066
13	217772	217797	217806	217847	217847
14	216802	216836	216868	216868	216868
15	213809	213859	213850	213846	213846
16	215013	215034	215086	215047	215071
17	217896	217903	217940	217911	217931
18	219949	219965	219984	219984	219984
19	214332	214341	214375	214346	214355
20	220833	220865	220899	220852	220882
21	304344	304351	304387	304350	304363
22	302332	302333	302379	302345	302379
23	302354	302408	302416	302416	302416
24	300743	300757	300757	300743	300747
25	304344	304344	304374	304357	304366
26	301730	301754	301836	301796	301836
27	304949	304949	304952	304949	304952
28	296437	296441	296478	296456	296466
29	301313	301331	301359	301357	301357
30	307014	307078	307089	307072	307089

Table C.3: Best known solutions found for $n = 500$ and $m = 30$

No	GA[11]	TS[57]	Fix+TS[58]	SAS/WO	SAS/W
1	115868	115991	116056	115845	115848
2	114667	114810	114810	114753	114753
3	116661	116683	116712	116614	116614
4	115237	115301	115329	115176	115249
5	116353	116435	116525	116289	116398
6	115604	115694	115741	115574	115634
7	113952	114003	114181	114072	114072
8	114199	114213	114348	114211	114223
9	115247	115288	115419	115197	115419
10	116947	117055	117116	116921	117020
11	217995	218068	218104	218008	218068
12	214534	214562	214648	214518	214626
13	215854	215903	215978	215807	215918
14	217836	217910	217910	217760	217801
15	215566	215596	215689	215591	215594
16	215762	215842	215890	215743	215867
17	215772	215838	215907	215831	215879
18	216336	216419	216542	216419	216419
19	217290	217305	217340	217209	217313
20	214624	214671	214739	214627	214628
21	301627	301643	301675	301643	301656
22	299985	300055	300055	299951	299985
23	304995	305028	305087	305021	305062
24	301935	302004	302032	301957	301970
25	304404	304411	304462	304391	304391
26	296894	296961	297012	296902	296949
27	303233	303328	303364	303328	303328
28	306944	306999	307007	306885	306944
29	303057	303080	303199	303122	303147
30	300460	300532	300572	300484	300516