Achieving Integer Feasibility Quickly by Alternating Axis-Parallel and General Disjunctions

by

Hanan Mahmoud, B. Eng.

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs

in partial fulfillment of the requirements for the degree of

Master of Applied Science in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE)

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

January 2012

© Copyright 2012, Hanan Mahmoud

Library and Archives
Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 978-0-494-87801-9

Our file Notre référence

ISBN: 978-0-494-87801-9

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

The undersigned recommend to
the Faculty of Graduate and Postdoctoral Affairs
acceptance of the thesis,

**Achieving Integer Feasibility Quickly by
Alternating Axis-Parallel and General Disjunctions**

submitted by

Hanan Mahmoud, B. Eng.

in partial fulfillment of the requirements for
the degree of Master of Applied Science in Electrical and Computer Engineering

Chair, Howard Schwartz, Department of Systems and Computer Engineering

Thesis Supervisor, John W. Chinneck

Carleton University

January, 2012

Abstract

Many real world problems can be cast as Mixed-Integer Linear Programs (MIPs) in which the constraints are linear and some or all of the variables are required to have integer or binary values. Branch and Bound (B&B) is a well-known method for solving MIPs. Most B&B algorithms consider only a small set of branching disjunctions that are based on selecting a branching variable to create an axis-parallel disjunction. Although such variable disjunctions can be effective, they limit the focus to a single integer-constrained variable at a time. On the other hand, general disjunctions branch on multiple integer-constrained variables at once and thus can speed the MIP solution. This thesis studies the use of both axis-parallel and general disjunctions in the same MIP solution to speed the achievement of the first MIP-feasible solution. The general themes of the research are (i) using a larger set of disjunctions than the usual axis-parallel branching, (ii) identifying when to change the type of disjunction in use, and (iii) ways of constructing useful general disjunctions. Comparative results for Cplex, a state-of-the-art commercial solver, show significant reductions in the time to find the first MIP-feasible solution. This demonstrates the power of alternating between axis-parallel and general disjunctions to quickly achieve integer feasibility.

Acknowledgements

I would like to sincerely thank my supervisor Professor John Chinneck for his continuous support, guidance and encouragement. His promptness and continual advice have helped me throughout my research. His everlasting energy and passion for research in science and engineering has been an influential example for myself, as well as others. It is because of him I find research very rewarding.

I would like to also thank the many people who made this research possible. Special thanks to the Systems and Computer Engineering Department staff who were always there to provide administrative and technical assistance.

My deepest gratitude goes to my beloved family who have been a perpetual source of support and encouragement throughout the years. I am indebted to my mother, Dr. Iman, and my father, Dr. Akram, for their immense support and enduring love which have enabled me to complete my research. I would also like to genuinely thank my brothers for their endless patience, and emotional support that helped me get through the most difficult times of my studies.

Table of Contents

Abstract	iii
Acknowledgements	iv
Table of Contents	v
List of Tables	vii
List of Figures	viii
1. Introduction	1
2. Mixed Integer Linear Programming	4
3. Seeking MIP Feasibility: State of the Art.....	10
3.1. Finding the First MIP-Feasible Solution Quickly	10
3.2. General Disjunctions	12
4. Problem Statement	15
5. New Branching Heuristics Using General Disjunctions	17
5.1. When to Include a General Disjunction	17
5.2. Introduction to 45-Degree General Disjunctions.....	24
5.3. Choosing an Active Constraint for Branching	27
5.4. Forming a 45-Degree General Disjunction Based on an Active Constraint	36
5.4.1. Creating a Perpendicular General Disjunction	38
5.5. Branching Direction of General Disjunctions	43
5.5.1. More on Branching Direction Selection Schemes for Parallel General Disjunctions.....	46
6. Experimental Setup	50
6.1. Software	50
6.1.1 Programming Language and Compiler	50
6.1.2 Cplex Parameter Settings	50
6.1.3 Programming the New Algorithms.....	52
6.2. Hardware.....	53
6.3. Test Models.....	53
6.4. Metrics.....	56
7. Designing the Best Algorithm	60

7.1. Experiment 1: Deciding When to Perform a General Disjunction	61
7.1.1 Recognizing That Axis-Parallel Disjunctions are Performing Poorly ..	62
7.1.2. Limiting the Number of General Disjunctions	67
7.1.3 Adding a General Disjunction After the Root Node	78
7.2. Experiment 2: Choosing an Active Constraint for General Branching	81
7.2.1. A Closer Look at the Number of Tied Branching Constraints	86
7.3. Experiment 3: Creating a 45-Degree Perpendicular General Disjunction.	88
7.4. Experiment 4: Selecting the Branching Direction For General Disjunctions	96
7.4.1. Experiment 4.A: Selecting the Branching Direction and Offset For Parallel General Disjunctions	97
7.4.2. Experiment 4.B: Selecting the Branching Direction For Perpendicular General Disjunctions	102
7.5. Experiment 5: Constructing the Best Algorithms	105
8. Comparison to the State of the Art	110
8.1. Comparing the Top Algorithms Against Baseline_Cplex Over the Hard MIPs	110
8.2. Overall Performances of Top Algorithms	115
9. Conclusions and Future Work	121
9.1. Conclusions	121
9.2. Summary of Contributions	122
9.3. Future Work	123
10. References	126
A.1 MIPLIB 2010 Subset of Equality-Containing MIP Models	128
A.2 MIPLIB 2010 Subset of Inequality-Containing MIP Models	132
A.3 MIPLIB 2010 Subset of Equality and Inequality-Containing MIP Models ...	136
A.4 MIPLIB 2010 Subset of Easy MIP Models Used for Testing	140
A.5 MIPLIB 2010 Subset of Hard MIP Models Used for Testing	144
A.6 MIPLIB 2010 Subset of Equality-Containing MIP Models Using Perpendicular General Disjunctions	148

List of Tables

Table 1: Example of computing the count and ratio measures.	20
Table 2: Summary of branching-constraint selection algorithms.	30
Table 3: Default general branching algorithm used in all training experiments...	61
Table 4: Summary of methods for triggering the inclusion of a general disjunction.	63
Table 5: Simplex iterations of models that are executed relatively quickly by the best methods that trigger the inclusion of a general disjunction.	68
Table 6: Summary of branching constraint selection methods compared in Experiment 2.	82
Table 7: Average percentage of tied top-ranked active constraints for each branching-constraint selection method.	87
Table 8: Summary of least-impact-variable-selection methods compared in Experiment 3.	90
Table 9: Summary of branching-direction selection methods compared in Experiment 4.A.	97
Table 10: Summary of top methods for making each decision in a complete algorithm.....	106
Table 11: Percentages of models solved by each algorithm for all 55 hard MIP models.....	111
Table 12: Summary of results for each of the top algorithms in comparison to <i>Baseline_Cplex</i>	115
Table 13: Comparing the number of backtracks performed by each of the top algorithms versus <i>Baseline_Cplex</i>	119
Table 14: Comparing the optimality gap of each of the top algorithms versus <i>Baseline_Cplex</i>	120

List of Figures

Figure 1: Graphical representation of a MIP model.	5
Figure 2: Creating child nodes for a candidate variable.	7
Figure 3: Child nodes of a general disjunction.	25
Figure 4: Approximately parallel 45-degree general disjunction.	26
Figure 5: Branching parallel to a 45-degree equality constraint.	37
Figure 6: Branching in the satisfying or the violating direction for the branching constraint.	47
Figure 7: Experiment 1 performance profile on computational time for detecting when axis-parallel disjunctions are performing poorly.	65
Figure 8: Experiment 1 performance profile on simplex iterations for detecting when axis-parallel disjunctions are performing poorly.	66
Figure 9: Experiment 1 performance profile on node count for detecting when axis-parallel disjunctions are performing poorly.	66
Figure 10: A histogram of the number of general disjunctions triggered by $(CVC+ISC)>5$ at different levels in the B&B tree for 30n20b8.mps, gmut-77-40.mps, and csched007.mps.	69
Figure 11: A histogram of number of general disjunctions triggered by $(CVC+ISC)>10$ at different levels in the B&B tree for 30n20b8.mps, gmut-77-40.mps, and csched007.mps.	69
Figure 12: A histogram of the number of general disjunctions triggered by $(CVC-And-ISC)>3$ at different levels in the B&B tree for acc-tight5.mps, gmut-75-50.mps, neos-1171737.mps, and mzzv11.mps.	70
Figure 13: A histogram of the number of general disjunctions triggered by $(CVC-Or-ISC)>10$ at different levels in the B&B tree for gmut-77-40.mps, b2c1s1.mps, and lotsize.mps.	70
Figure 14: A histogram of the number of general disjunctions triggered by $(CVC-And-ISC)>3$ for the models on which it performs poorly.	71
Figure 15: A histogram of the number of general disjunctions triggered by $(CVC-Or-ISC)>10$ for the models on which it performs poorly.	72
Figure 16: Experiment 1 performance profile on simplex iterations for determining the best global requirement on the minimum number of candidate variables for $(CVC+ISC)>5$	74
Figure 17: Experiment 1 performance profile on simplex iterations for determining the best global requirement on the minimum number of candidate variables for $(CVC+ISC)>10$	74
Figure 18: Experiment 1 performance profile on simplex iterations for determining the best global requirement on the minimum number of candidate variables for $(CVC-And-ISC)>3$	75

Figure 19: Experiment 1 performance profile on simplex iterations for determining the best global requirement on the minimum number of candidate variables for $(CVC-Or-ISC)>10$.	77
Figure 20: Experiment 1 performance profile on simplex iterations for determining the best method for triggering the inclusion of a general disjunction with a global requirement on the minimum number of candidate variables.	78
Figure 21: Experiment 1 performance profile on computational time for comparing the performance of $(CVC-And-ISC)>3$ with and without adding a general disjunction at the root node.	79
Figure 22: Experiment 1 performance profile on simplex iterations for comparing the performance of $(CVC-And-ISC)>3$ with and without adding a general disjunction at the root node.	80
Figure 23: Experiment 1 performance profile on node count for comparing the performance of $(CVC-And-ISC)>3$ with and without adding a general disjunction at the root node.	80
Figure 24: Experiment 2 performance profile on time for choosing an active constraint for general branching.	85
Figure 25: Experiment 2 performance profile on simplex iterations for choosing an active constraint for general branching.	85
Figure 26: Experiment 2 performance profile on node count for choosing an active constraint for general branching.	86
Figure 27: Experiment 3 performance profile on time for choosing the variable with the least impact on the MIP solution.	93
Figure 28: Experiment 3 performance profile on simplex iterations for choosing the variable with the least impact on the MIP solution.	93
Figure 29: Experiment 3 performance profile on node count for choosing the variable with the least impact on the MIP solution.	94
Figure 30: Experiment 3 performance profile on simplex iterations for choosing the variable with the least impact on the MIP solution (for only models that used perpendicular general disjunctions).	95
Figure 31: Experiment 3 performance profile on simplex iterations for creating a 45-degree perpendicular disjunction using <i>SC-var</i> by switching signs on odd or even counts.	96
Figure 32: Experiment 4.A performance profile on time for selecting the branching direction for parallel general disjunctions.	99
Figure 33: Experiment 4.A performance profile on simplex iterations for selecting the branching direction for parallel general disjunctions.	99
Figure 34: Experiment 4.A performance profile on node count for selecting the branching direction for parallel general disjunctions.	100

Figure 35: Experiment 4.A performance profile on simplex iterations for comparing the performance of <i>Sat-BC</i> with and without offsetting the RHS of the disjunction by one unit.	101
Figure 36: Experiment 4.B performance profile on time for selecting the branching direction for perpendicular general disjunctions.	104
Figure 37: Experiment 4.B performance profile on simplex iterations for selecting the branching direction for perpendicular general disjunctions.	104
Figure 38: Experiment 4.B performance profile on node count for selecting the branching direction for perpendicular general disjunctions.	105
Figure 39: Experiment 5 performance profile on time for determining the best combinations of algorithms.	108
Figure 40: Experiment 5 performance profile on simplex iterations for determining the best combinations of algorithms.	108
Figure 41: Experiment 5 performance profile on node count for determining the best combinations of algorithms.	109
Figure 42: Performance profile on time for comparing the top complete algorithms against <i>Baseline_Cplex</i> for the hard MIP models.	112
Figure 43: Performance profile on simplex iterations for comparing the top complete algorithms against <i>Baseline_Cplex</i> for the hard MIP models.	113
Figure 44: Performance profile on node count for comparing the top complete algorithms against <i>Baseline_Cplex</i> for the hard MIP models.	113
Figure 45: Performance Profiles on simplex iterations comparing each of the top 4 methods against <i>Baseline_Cplex</i>	114
Figure 46: Performance profile on simplex iterations for comparing the top complete algorithms against <i>Baseline_Cplex</i> for all types of MIP models.	117
Figure 47: Comparing the rate of decrease in number of candidate variables for <i>Baseline_Cplex</i> Vs "2.B, 3.B, 5.A" for <i>Satellites1-25</i>	118
Figure 48: Observing the rate of decrease in number of candidate variables and infeasibility sum at different nodes for "2.B, 3.B, 5.A" for <i>Satellites1-25</i>	119

1. Introduction

A Mixed-Integer Programming (MIP) problem is a linear program in which some or all of the variables are required to take on integer or binary values [Chen et al. 2009]. A MIP problem is typically composed of an objective function that is maximized or minimized, and is subject to general linear constraints, as well as integrality restrictions and variable bounds. A classical MIP problem would take the following form:

$$\begin{aligned} \text{Max/ Min } Z &= \mathbf{c}_1\mathbf{x} + \mathbf{c}_2\mathbf{y} \\ \text{Subject to} \\ \mathbf{A}_1\mathbf{x} + \mathbf{A}_2\mathbf{y} &\{\geq, \leq, =\} \mathbf{b} \\ \mathbf{x} &\in \mathbb{R}^d, \mathbf{y} \in \mathbb{Z}^{n-d} \\ \mathbf{l}_1 &\leq \mathbf{x} \leq \mathbf{u}_1, \mathbf{l}_2 \leq \mathbf{y} \leq \mathbf{u}_2, \end{aligned}$$

where n is the total number of variables such that $\mathbf{x}$ is a vector of d continuous variables ($x_1, x_2, \dots, x_d$), $\mathbf{y}$ is a vector of $n-d$ integer-constrained variables ($y_1, y_2, \dots, y_{n-d}$), $\mathbf{c}_1$ and $\mathbf{c}_2$ are defined coefficient vectors, $\mathbf{A}_1$ and $\mathbf{A}_2$ are defined coefficient matrices, $\mathbf{b}$ is a well-defined vector of right-hand-side values, $\mathbf{l}_1, \mathbf{u}_1$ and $\mathbf{l}_2, \mathbf{u}_2$ are known vectors that represent the lower and upper bounds for the $\mathbf{x}$ and $\mathbf{y}$ variables, respectively. Z is a variable that holds the maximized or minimized value of the objective function. The vectors $\mathbf{c}_1, \mathbf{c}_2, \mathbf{l}_1, \mathbf{l}_2, \mathbf{u}_1$, and $\mathbf{u}_2$ as well as the matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ can have real or integer values.

MIPs can be used to model a number of different real-world discrete optimization problems. They are commonly, but not exclusively, used to model problems in transportation, staff scheduling, production planning, supply chain management, and project management [Chen et al. 2009].

Branch-and-Bound (B&B) is a well-known method that is commonly used for solving MIPs. While the optimum solution is usually the goal, finding the first MIP-feasible solution to the problem is a significant part of the process. A B&B algorithm typically consists of a number of essential decision-making heuristics that can greatly affect the speed and/or efficiency of the solution process. The node selection scheme, as well as the branching variable selection strategy and the branching direction, are all important choices while developing a B&B algorithm. Various heuristics for making these decisions are incorporated in modern MIP solvers; a number of these are mentioned in Sections 2 and 3.

Most solvers consider only a small set of branching disjunctions that focus on using branching variable-selection heuristics to choose the best integer variable for branching. Unfortunately, this limits the number of branching variables to only one per node in the B&B tree. Limited research has been conducted in the literature on the use of general disjunctions, which are disjunctions that consider multiple integer variables in a single disjunction equation. Indeed, there are no active studies that consider the use of general disjunctions for achieving MIP-feasibility quickly. This thesis will introduce new branching heuristics that make use of general disjunctions to attain the first feasible solution as quickly as possible.

Experiments that motivate the use of a variety of branching disjunctions are developed and compared against state-of-the-art B&B algorithms. Different methods of creating the general disjunctions, and ways of deciding the branching direction when they are used, are developed and empirically evaluated. All of

these new methods aim to obtain the first integer-feasible solution as quickly as possible. This is also likely to speed the attainment of the optimum solution as well, partly by providing an early incumbent for pruning the tree. The empirical results illustrate the potency of alternating between different kinds of branching disjunctions to achieve MIP-feasibility quickly. By creating an effective general disjunction, deciding on the right branching direction for the disjunction, and determining when to switch between the different types of disjunctions, these new algorithms prove to be better at finding the first integer-feasible solution faster than state-of-the-art solvers.

2. Mixed Integer Linear Programming

A MIP, otherwise known as a Mixed Integer Linear Program (MILP), is a linear program in which at least one variable is restricted to take on an integer or binary value. Such integrality restrictions, along with other general constraints, are required to be satisfied in order to obtain a MIP-feasible solution. An LP-feasible solution solely satisfies the general linear constraints and bounds of a MIP model, whereas a MIP-feasible (or integer-feasible) solution satisfies all problem constraints including the integrality restrictions. The term MIP can also be used to describe an LP in which all the variables are required to take on integer values, otherwise known as pure Integer Programming (IP). Binary Integer Programming (BIP) problems are also a sub-class of MIPs in which all variables are required to take on binary values (0 or 1). Hence, the term MIP represents all three kinds of problems (MIPs, IPs, and BIPs). Below is an example of a 2-dimensional IP model that is shown in Figure 1:

$$\text{Max } Z = -2y_1 + 5y_2$$

Subject to

$$15y_1 + y_2 \leq 33$$

$$-25y_1 + 5y_2 \leq 3,$$

where y_1 and y_2 are integer variables with variable bounds $y_1, y_2 \geq 0$.

Figure 1 below graphically represents the MIP model. The bold lines represent the two constraints, the dotted lines, l_1 and l_2 , represent the objective function at the LP-optimum and the MIP-optimum points respectively, and the dark dots in the middle represent the integer-feasible solutions within the constraints and variable bounds. Initially, the MIP problem is LP-feasible but MIP-

infeasible, as the y_1 and y_2 variables do not take on integer values at the LP-optimum point. The LP-relaxation solution (i.e. solution attained by relaxing the integrality constraints on integer variables) to the MIP model is represented in Figure 1 by the top corner point where the two general constraints and I_1 intersect. All LP-relaxation solutions lie at the corner points, which are defined by the *active* constraints, of the feasible region. A constraint is active if it is an equality or a tight inequality constraint at the current LP-relaxation point. At one point in the MIP-solving process, the adjusted variable bounds will create the LP-optimum corner point (1,5), and at this point, the MIP-optimum point is also reached, as this is the maximum objective function value that can be attained while satisfying the integrality restrictions on y_1 and y_2 .

Figure 1: Graphical representation of a MIP model.

Branch and Bound uses a divide-and-conquer approach in which the original model is divided into sub-models with tightened bounds that are progressively generated and solved [Chen et al. 2009]. The solution process is typically represented using a B&B tree, where each node of the tree corresponds to an LP sub-problem and each edge corresponds to the tightened bound on an integer variable.

The B&B algorithm begins by relaxing the integrality constraints of the MIP problem to form a conventional LP problem. The LP relaxation of the MIP is then solved using a regular LP-solver to obtain an LP-optimal solution. The LP-relaxation solution at the root node may not necessarily be feasible, which shows that the original MIP is infeasible. On the other hand, if there exists an LP-feasible solution, it may not be MIP-feasible if some, or all, integer variables are not assigned integer values. An integer variable that does not take on an integer value is called a *candidate variable*. At this point, a standard B&B method picks one of the candidate variables for branching. Two child nodes are created: the down branch is created by adjusting the upper bound on the branching variable to force it to a lower value, and the up branch adjusts the lower bound on the branching variable to force it to a higher value. The lower and upper bounds are adjusted in the child nodes as shown in Figure 2.

Figure 2: Creating child nodes for a candidate variable.

As noted in Figure 2, x_i is the branching variable, which is assigned a fractional value during the LP relaxation, and k is the rounded down value of x_i . The down child created in Figure 2 adds an upper bound on x_i , such that x_i is restricted to take on any value less than or equal to the rounded down value of the current value of x_i . On the other side, the up child adds a lower bound on x_i , such that x_i is restricted to take on any value greater than or equal to the rounded up value of the current value of x_i . Figure 2 illustrates the possible branching directions (up or down) for a candidate variable at any node in the B&B tree. For every node in the B&B tree, there are $2p$ possible branches, where p is the number of candidate variables at the current node.

To characterize the quality of a solution, the notion of *integrality gap* is sometimes used [Karamanov and Cornuéjols, 2011] [Cornuéjols, Liberti and Nannicini, 2011]. The integrality gap is defined as the maximum ratio between the optimum of a MIP solution and of its LP relaxation [Chlamtac and Tulsiani, 2011], as illustrated in the following equation for a maximization problem:

$$Integrality\ Gap = \frac{OPT_{LP}}{OPT_{MIP}}$$

where OPT_{LP} represents the optimal solution of the LP relaxation of the MIP model, and OPT_{MIP} represents the optimal MIP solution of the model. For a minimization problem, the following equation is used:

$$Integrality\ Gap = \frac{OPT_{MIP}}{OPT_{LP}}.$$

Choosing the right candidate variable for branching is another important decision that can significantly affect the speed and efficiency of the MIP solution. Various branching variable selection heuristics, such as estimating the impact of the different candidate variables on the objective function, exist [Patel and Chinneck, 2007]. A number of other heuristics, such as selecting the branching variable that maximizes the degradation of the objective function value, are highlighted in Section 3.

Since the algorithms in this thesis focus on achieving the first feasible solution, the process of selecting a candidate variable for branching is repeated only until a MIP-feasible solution is reached. A MIP problem is infeasible if all leaf nodes in the B&B tree are LP-infeasible. A node selection algorithm is required at any node to decide on the next node to explore in the tree. The most common node selection scheme for achieving quick feasibility in MIPs is depth-first. This is because it has the advantage of using a warm start from the LP relaxation solution of the parent node while diving down the tree to reach the first MIP-feasible leaf [Patel and Chinneck, 2007]. The B&B algorithm for finding the first feasible solution, using depth-first node-selection, is summarized below in Algorithm 1.

Input: MIP. List of unexplored nodes = ϕ .

1. Solve the LP relaxation of the root node (original MIP model)
 - 1.1 If the LP relaxation is infeasible, exit with infeasible outcome.
 - 1.2 If the LP relaxation is MIP-feasible, exit with LP relaxation solution as the solution to the original MIP.
2. Choose a candidate variable for branching
3. Create two child nodes from the current node (parent node) by tightening the upper and lower bounds of the candidate variable.
4. Add the child nodes to the list of unexplored nodes.
5. Using a branching-direction selection criterion, choose one of the just created child nodes to explore next. Remove chosen node from list of unexplored nodes.
6. Solve the LP relaxation of the selected node.
 - 6.1 If LP relaxation is infeasible, discard node and select the most recently-created node, according to the depth-first criteria, from the list of unexplored nodes.
 - 6.2 If the LP relaxation is MIP-feasible, exit with LP relaxation solution as the solution to the original MIP.
7. Go to Step 2.

Algorithm 1: B&B algorithm for finding the first feasible solution using depth-first node selection.

In addition to axis-parallel branching (i.e. disjunctions that branch on only one integer variable per node) which are used in Algorithm 1, the algorithms in this thesis also use general disjunctions. These disjunctions branch on multiple integer variables as they take the form $a_1x_1 + a_2x_2 + \dots + a_nx_n \leq k$ and $a_1x_1 + a_2x_2 + \dots + a_nx_n \geq k+1$ where the a_i are integer constants and the x_i are variables. Section 5 of the thesis provides more details on the general disjunctions used in the algorithms developed here.

3. Seeking MIP Feasibility: State of the Art

Most MIP studies in the literature focus on obtaining the optimal solution to the problem, yet achieving MIP-feasibility quickly is fundamental to the process. If optimality is the goal, then finding a feasible solution quickly will allow early pruning of nodes in the B&B tree, hence attaining a smaller search tree. On the other hand, if feasibility is the goal, then finding a feasible solution quickly is directly useful [Patel and Chinneck, 2007]. Furthermore, once a MIP-feasible node is reached in the B&B tree, one can make use of smart node selection rules [Wojtaszek and Chinneck, 2010] to determine the next node to explore to efficiently seek optimality.

3.1. Finding the First MIP-Feasible Solution Quickly

Fischetti et al. [2004] considered the *NP*-hard problem of reaching MIP-feasibility for a general MIP problem. They suggest a Feasibility Pump (FP) heuristic based on a distance function defined between an LP solution x^* and an integer-rounded point $\tilde{x}$. The FP heuristic alternates between the optimum solution to the LP $\min\{\Delta(x, \tilde{x}) : x \in P\}$, where P is defined as $\{x : Ax \geq b\}$ for a generic MIP problem of the form $\min\{c^T x : Ax \geq b, x_j \text{ integer } \forall j \in \text{set of integer variables}\}$. This minimizes the distance between the current LP solution x^* and its rounded solution $\tilde{x}$. The LP problem is solved at each FP iteration until a feasible MIP solution is achieved, i.e. a distance of zero is attained between x^* and $\tilde{x}$. The authors report good computational results on binary MIP problems, which makes FP a useful root

node heuristic for seeking MIP-feasibility. However, unlike B&B, finding a MIP-feasible solution using this method is not guaranteed.

Patel and Chinneck [2007] presented a new approach towards finding the first MIP-feasible solution as quickly as possible. They estimate the impact of candidate variables on the active constraints in the LP relaxation. Their algorithms start off by assigning weights to the candidate variables according to a specific criterion, and then select the variable with the highest weight as the branching variable. The authors suggested a number of variable selection schemes based on measuring the influence of a candidate variable within an active constraint and/or measuring the extent to which an active constraint could be influenced by a variable.

Patel and Chinneck [2007] report empirical results that show great improvements over Cplex, a state-of-the-art commercial solver. Their most successful branching variable selection method, *Method A*, chooses the branching variable that appears in the largest number of active constraints and excels at finding the first feasible solution very quickly. The authors conclude that the use of active constraints to identify the branching variable shows substantial promise towards achieving MIP-feasibility quickly. Some of the algorithms in this thesis make use of some of the core ideas in their active constraint-based variable selection schemes.

Pryor and Chinneck [2011] recently developed a number of new branching variable and branching direction selection methods to reach MIP-feasibility faster. Given a variable/direction pair, they estimate the probability of satisfying a

constraint at the child node. The authors surprisingly conclude that the child with the smallest probability is most likely to lead to the first integer-feasible solution more quickly. This is because the lower-probability choice forces the greatest number of candidate variables to change concurrently. The authors present empirical results which show a strong correlation between probability-based branching methods and achieving MIP-feasibility faster. They also emphasized on the role of "*multiple-choice*" constraints, which are composed entirely of binary variables and have the form $x_1 + x_2 + x_3 + \dots + x_n \{\leq, =\} 1$, on achieving integer-feasibility quickly. They conclude that branching up on a variable in a multiple-choice constraint is most desirable for reaching MIP-feasibility quickly as it forces all variables in the constraint to take on integer values. Additional background information on seeking MIP-feasibility in the literature can also be found in [Chinneck 2008].

3.2. General Disjunctions

Branching is an essential part of the B&B method; hence selecting the right branching disjunction can greatly affect performance. This section discusses the literature on the use of general disjunctions to achieve an optimal MIP solution. In contrast, algorithms in this thesis use general disjunctions from a feasibility perspective rather than optimality.

Mahajan and Ralphs [2009] conducted a number of computational experiments to study the effects of general disjunctions. Their objective was to find a pair (π, π_0) for a general disjunction of the form $\pi x \leq \pi_0$ or $\pi x \geq \pi_0 + 1$, where $(\pi, \pi_0) \in \mathbb{Z}^d \times 0^{n-d} \times \mathbb{Z}$ such that the new lower bound $z_l = \min \{z_L^*, z_R^*\}$ is

maximized for a minimization MIP problem consisting of d integer variables and $n-d$ real variables. z_L^* represents the objective function value of the LP sub-problem at the left child node containing the general disjunction constraint $\pi x \leq \pi_0$; whereas z_R^* represents the objective function value of the LP sub-problem at the right child node containing the general disjunction constraint $\pi x \geq \pi_0 + 1$. The authors used two criteria to evaluate the branching disjunction: (i) lower bound improvement achieved after branching and (ii) width of the created LP sub-problem in the direction of the disjunction. They were able to improve the overall efficiency in terms of reducing the number of nodes in the search tree. In a related study [Mahajan and Ralphs, 2010], they determined that the complexity of selecting general disjunctions at each iteration of the B&B algorithm is *NP*-hard. However, they also stated that selecting a general disjunction in which all the coefficients of the disjunction function are in the set $\{-1, 0, 1\}$ results in an *NP*-complete problem. Consequently, these studies motivate the idea of developing fast heuristics to create effective general disjunctions.

Karamanov and Cornuéjols [2011] also realized that branching on general disjunctions could result in a smaller tree size as they more often lead to only one feasible child node instead of two. To obtain an efficient algorithm, Karamanov and Cornuéjols only considered disjunctions that define Gomory Mixed Integer Cuts (GMICs) where the distance cut off by the intersection is used to measure the quality of the disjunction. They also recognized that using general disjunctions closes more of the integrality gap.

Cornuéjols, Liberti and Nannicini [2011] reconfirmed that general branches can reduce the number of nodes in the enumeration tree by a factor of more than two. They constructed split disjunctions arising from GMICs generated from linear combinations of the rows of the simplex tableau. The authors considered a general disjunction to be profitable if it generates the smallest number of feasible children or if it closes more of the integrality gap. They conducted computational experiments combining single-variable disjunctions and general disjunctions, which led to an increase in closed gap per node but did not translate to computational time reductions.

Owen and Mehrotra [2001] proposed branching on general disjunctions generated by a neighbourhood search heuristic. The neighbourhood heuristic uses disjunctions with coefficients only from the set $\{-1, 0, 1\}$ on integer variables at the current node of the B&B tree. Such disjunctions are oblique with an angle of 45 degrees, hence they only pass through integer-feasible points in the feasible region. The heuristic search procedure aims at finding a disjunction that maximizes the value of the disjunction over all possible valid disjunctions. The authors reach the conclusion that such general disjunctions could significantly decrease the size of the branching tree. However, their experiments were designed to isolate the effects of branching on general disjunctions, so they did not focus on the time requirements of their algorithms and thus did not report any time improvements.

4. Problem Statement

Finding a MIP-feasible solution as quickly as possible can be very useful as it allows the early pruning of subsequent feasible nodes in the B&B tree. In addition, achieving MIP-feasibility quickly may be all that is required for some applications. As a result, the main goal of this thesis is to develop heuristics that allow us to reach the first feasible solution of a MIP as quickly as possible.

Most solvers consider only disjunctions based on selecting a branching variable to create an *axis-parallel disjunction*. These typically take the form $x_i \leq k$ and $x_i \geq k+1$ for some candidate variable x_i and the largest integer k that is less than x_i . These variable disjunctions focus on only one integer variable at a time; hence more effort is put into choosing the right branching variable and direction [Pryor and Chinneck, 2011].

The goal of this thesis, i.e. achieving MIP-feasibility quickly, will be accomplished by expanding on the set of disjunctions used in the B&B tree. Since no studies in the literature use general disjunctions for the purpose of attaining MIP-feasibility quickly, this thesis explores the idea of alternating traditional axis-parallel disjunctions and more advanced general disjunctions to obtain the first integer-feasible solution as quickly as possible. As stated in Section 3, Cornuéjols, Liberti and Nannicini [2011] conducted experiments that combine single-variable disjunctions with general disjunctions; however, their performance with respect to reaching MIP-feasibility quickly is unknown. Algorithms developed here will mainly use *45-degree* general branches similar to those used by Owen and Mehrotra [2001]. This term is used in this thesis to

represent general disjunctions that are restricted to the coefficients of +1, -1, and 0.

Using ideas on the importance of active constraints in reaching MIP-feasibility quickly [Patel and Chinneck 2007], the 45-degree general branches are constructed to be as parallel or as perpendicular as possible to an active constraint in the LP relaxation solution. Moreover, various metrics that monitor the performance of the branching algorithm are used to determine when to switch between the different kinds of branching disjunctions. The performance of the diverse algorithms developed here are compared and evaluated against each other and against the default B&B algorithm used in the well-known commercial solver Cplex 12.1 [Ilog 2009a, 2009b, 2009c].

5. New Branching Heuristics Using General Disjunctions

This thesis explores new fast MIP-feasibility heuristics involving the use of a wider set of disjunctions. General disjunctions, which take the form $a_1x_1 + a_2x_2 + \dots + a_nx_n \leq k$ and $a_1x_1 + a_2x_2 + \dots + a_nx_n \geq k+1$ where the a_i are integer constants and the x_i are variables, are used in the algorithms developed here. Various important aspects of the developed algorithms, such as when and how general disjunctions are used, are discussed in this section.

5.1. When to Include a General Disjunction

Most researchers caution that the effort required to generate general disjunctions and use them in the LP relaxations at all descendent nodes, outweighs the gains in the overall efficiency of the algorithm [Mahajan and Ralphs, 2009]. There is a trade-off between using axis-parallel disjunctions and general disjunctions. General disjunctions have the advantage of forcing multiple candidate variables to change at once, as well as more often leading to a smaller tree size [Owen and Mehrotra, 2001] [Mahajan and Ralphs, 2009] [Cornuéjols, Liberti and Nannicini, 2011] [Karamanov and Cornuéjols, 2011]. Axis-parallel disjunctions are also useful as such disjunctions only change the bounds on candidate variables, which does not increase the size of the MIP model and thus generate LP sub-problems at the child nodes that are easier to re-optimize. On the contrary, branching on a general disjunction adds an extra constraint to the LP sub-problems created at the child nodes, and all of their descendents. If general disjunctions are used frequently enough, then the size of the MIP model can increase remarkably (due to added general constraints) leading to an

increased solution time for each sub-problem in the B&B tree, and hence a decrease in the overall efficiency of the algorithm [Karamanov and Cornuéjols, 2011].

Given that the over-use of general disjunctions to solve MIPs can counteract their gains, the algorithms developed here use axis-parallel disjunctions as the default type of disjunction, while occasionally using general disjunctions when it is advantageous to do so. There is a question of cost-benefit analysis in determining when the benefits of a general disjunction offset the added cost. The algorithms developed here generally consider two questions when deciding whether to insert a general disjunction: (i) has the solution progress stalled while using only axis-parallel branching? and (ii) are there sufficient candidate variables to make a general disjunction worthwhile?

The solution progress while using axis-parallel disjunctions is constantly monitored using two interrelated metrics: (i) the number of candidate variables, and (ii) the *infeasibility sum* at the current LP relaxation optimum. The infeasibility sum is the cumulative sum of the difference between the current value of each candidate variable and its closest integer value over all candidate variables at the current LP relaxation optimum [Ilog 2009c]. If either or both of these measures increase a number of times in a row while branching using axis-parallel disjunctions, then this is an indication that axis-parallel disjunctions are no longer being effective, hence a single general disjunction is generated to boost the performance of the B&B algorithm. Counting the increases and decreases in such measures is one way to determine whether the solver is getting stuck.

Another way is to observe the increase and decrease in the ratios of these measures between the parent and child nodes in the B&B tree.

A number of different algorithms are developed using the change in count or ratio of candidate-variable and infeasibility-sum measures. The candidate-variable count measure (*cand-var-count*) is incremented by 1 whenever the current number of candidate variables at a child node increases from the previous number of candidate variables at the parent node. Similarly, *cand-var-count* is decremented by 1 whenever the current number of candidate variables at a child node decreases from its previous value at the parent node. The same concept applies to the infeasibility-sum count measure (*infeas-sum-count*). On the other hand, the candidate-variable ratio measure (*cand-var-ratio*) captures the ratio of change in candidate variables between the parent node and the child node. The candidate-variable ratio-progress measure (*cand-var-ratio-progress*) is then used to capture the cumulative change in the value of *cand-var-ratio* in the B&B tree. The infeasibility-sum ratio (*infeas-sum-ratio*) and infeasibility-sum ratio-progress (*infeas-sum-ratio-progress*) measures are used in a similar manner to monitor the change in ratio of the infeasibility sum in the B&B tree. For example, given the number of candidate variables and infeasibility sum at a parent node and each of the child nodes, one can compute the values for the different types of measures as demonstrated in Table 1. This example assumes that all measures are initialized to 0. The formula for computing *cand-var-ratio* as well as a small example will be provided later in this section.

Table 1: Example of computing the count and ratio measures.

	Number of Candidate variables	Infeasibility Sum	<i>cand- var- count</i>	<i>infeas- sum- count</i>	<i>cand- var- ratio- progress</i>	<i>infeas-sum- ratio- progress</i>
Parent Node	83	232.5	--	--	--	--
Child 1 Node	70	202	$0 + (-1)$ = -1	$0 + (-1)$ = -1	$0 + \frac{70 - 83}{83}$ = -0.16	$0 + \frac{202 - 232.5}{232.5}$ = -0.13
Child 2 Node	95	257.5	$0 + 1$ = 1	$0 + 1$ = 1	$0 + \frac{95 - 83}{83}$ = 0.14	$0 + \frac{257.5 - 232.5}{232.5}$ = 0.11

Algorithm 2 illustrates how *cand-var-count* and *infeas-sum-count* can be used in one way to monitor the progress of the MIP solution. As seen in Algorithm 2, when the sum of both counts exceeds a certain threshold, a general disjunction is generated to boost the performance of the MIP solution. Using the sum of both counts indicates that the number of candidate variables and the infeasibility sum must both be increasing as we descend the B&B tree, and exceed a certain threshold in order for a general disjunction to be generated.

This algorithm is used whenever branching is required.

Input: MIP, *threshold* for triggering the use of a general disjunction.

1. If $(\text{cand-var-count} + \text{infeas-sum-count}) > \text{threshold}$ then
 - 1.1. Generate a general disjunction
 - 1.2. $\text{cand-var-count} \leftarrow 0$
 - 1.3. $\text{infeas-sum-count} \leftarrow 0$
2. Else: perform an axis-parallel branch
 - 2.1. If number of candidate variables increases from parent node then
 - 2.1.1. $\text{cand-var-count} \leftarrow \text{cand-var-count} + 1$
 - 2.2. Else : $\text{cand-var-count} \leftarrow \text{cand-var-count} - 1$
 - 2.3. If infeasibility sum increases from parent node then
 - 2.3.1. $\text{infeas-sum-count} \leftarrow \text{infeas-sum-count} + 1$
 - 2.4. Else : $\text{infeas-sum-count} \leftarrow \text{infeas-sum-count} - 1$

Algorithm 2: Using the sum of *cand-var-count* and *infeas-sum-count* to monitor the progress of the MIP solution and trigger the inclusion of a general disjunction.

At the start of the MIP solution, *cand-var-count* and *infeas-sum-count* are initialized to zero. A *threshold* measure is also initialized to a specific value, which indicates how frequent a general disjunction should be triggered. As seen in Algorithm 2, we are only interested in the increases in the number of candidate variables and/or infeasibility sum. The *cand-var-count* and *infeas-sum-count* measures decrease if the number of candidate variables and/or infeasibility sum stay the same or decrease as we descend the B&B tree.

Another way to monitor the progress of the MIP solution using *cand-var-count* and *infeas-sum-count* is given in Algorithm 3. This time both or either of the *cand-var-count* or *infeas-sum-count* must remain the same or increase a number of times in a row in order for a general disjunction to be generated. This algorithm makes sure that a general disjunction is only generated when the solution

progress is deteriorating or no longer improving for a number of times in a row down the B&B tree.

This algorithm is used whenever branching is required.

Input: MIP, *threshold* for triggering the use of a general disjunction.

1. If (*cand-var-count* > *threshold*) and/or (*infeas-sum-count* > *threshold*) then
 - 1.1. Generate a general disjunction
 - 1.2. *cand-var-count* $\leftarrow$ 0
 - 1.3. *infeas-sum-count* $\leftarrow$ 0
2. Else: perform an axis-parallel branch
 - 2.1. If number of candidate variables remains the same or increases from parent node then
 - 2.1.1. *cand-var-count* $\leftarrow$ *cand-var-count* + 1
 - 2.2. Else: *cand-var-count* $\leftarrow$ 0
 - 2.3. If infeasibility sum remains the same or increases from parent node then
 - 2.3.1. *infeas-sum-count* $\leftarrow$ *infeas-sum-count* + 1
 - 2.4. Else: *infeas-sum-count* $\leftarrow$ 0

Algorithm 3: Using the increase of *cand-var-count* and/or *infeas-sum-count* in a row to monitor the progress of the MIP solution and trigger the inclusion of a general disjunction.

Algorithm 4 illustrates yet another way to monitor the progress of the MIP solution, but this time using the candidate-variable and infeasibility-sum ratio measures (*cand-var-ratio* and *infeas-sum-ratio*). Algorithm 4 assumes that *cand-var-ratio* and *infeas-sum-ratio* are initialized to zero at the start of the program. The ratio measures capture the relative increase or decrease in the number of candidate variables and the infeasibility sum from one node to another down the B&B tree. For example, if the number of candidate variables at the parent node had a value of 50, and the number of candidate variables at the child node had a value of 40, then the candidate-variable ratio measure is calculated as follows:

cand-var-ratio =

$$\frac{(\text{Num. of cand. vars. at child node}) - (\text{Num. of cand. vars. at parent node})}{\text{Num. of cand. vars. at parent node}}$$

$$\begin{aligned}
&= \frac{40 - 50}{50} \\
&= -0.2,
\end{aligned}$$

which indicates that the number of candidate variables at the child node has decreased by 20% from its previous value at the parent node. Monitoring the change in ratios using *cand-var-ratio-progress* and *infeas-sum-ratio-progress* measures can help monitor the overall branching performance of the axis-parallel disjunctions, and hence determine when a general disjunction should be inserted. Once again, Algorithm 4 assumes that *cand-var-ratio-progress* and *infeas-sum-ratio-progress* are initialized to zero at the start of the program.

This algorithm is used whenever branching is required.

Input: MIP, *threshold* for triggering the use of a general disjunction.

1. If (*cand-var-ratio-progress* > *threshold*) and (*infeas-sum-ratio-progress* > *threshold*) then
 - 1.1. Generate a general disjunction
 - 1.2. Reset *cand-var-ratio-progress* and *infeas-sum-ratio-progress* to zero.
2. Else: perform an axis-parallel branch
 - 2.1. *cand-var-ratio* = (Current num. of candidate vars. – num. of cand. vars. at parent node) / (num. of cand. vars. at parent node)
 - 2.2. *cand-var-ratio-progress* = *cand-var-ratio-progress* + *cand-var-ratio*
 - 2.3. *infeas-sum-ratio* = (Current infeasibility sum – infeasibility sum at parent node) / (infeasibility sum at parent node)
 - 2.4. *infeas-sum-ratio-progress* = *infeas-sum-ratio-progress* + *infeas-sum-ratio*

Algorithm 4: Using ratio measures to monitor the progress of the MIP solution and trigger the inclusion of a general disjunction.

In addition to detecting when axis-parallel branching is performing poorly, there must be enough candidate variables at the current node to make a general disjunction worth generating. If the number of candidate variables at a particular node is fairly small, then the cost of inserting and maintaining a general

disjunction might outweigh the benefits of using it. Consequently, the algorithms developed here put a global requirement on the number of candidate variables before a general disjunction is considered. If the number of candidate variables at any node in the B&B tree is smaller than the limit, even if the solution progress has stalled while using axis-parallel branching, then a general branch will not be created at that node. The cut-off requirement along with any other conditions specified by the algorithms restrict the number of general disjunctions that are generated. By doing so, one can benefit from the occasional use of general disjunctions without incurring a very large overhead cost.

Deciding on the type of disjunction to use at the root node is another fundamental decision that can affect the speed of the MIP solution. Experiments in Section 7 compare the performance of the B&B algorithm with and without adding a general disjunction at the root node. A general disjunction can be used at the root node instead of an axis-parallel disjunction if and only if the number of candidate variables at the root node LP relaxation solution satisfies the cut-off requirement mentioned earlier.

5.2. Introduction to 45-Degree General Disjunctions

Mahajan and Ralphs' [2009, 2010] studies on general disjunctions in MIPs motivated the idea of creating efficient disjunctions where the effort necessary to generate such disjunctions does not counterweigh the overall gains of the algorithm. As a result, the algorithms developed here only consider "45-degree" general disjunctions, a special set of disjunctions. 45-degree general disjunctions are oblique branches in which the coefficients of the branching equation are

limited to the values 0, -1 or 1. Unlike common general disjunctions where there are no restrictions on the coefficients of the branching equation, 45-degree general disjunctions produce a clean separation where the edges of the disjunction lie on integer values only and there are no integer solutions between the pair of inequalities, as shown in Figure 4.

Since the main aim is to develop fast MIP-feasibility heuristics, ideas from Patel and Chinneck [2007] on the impact of active constraints are used to guide the creation of the 45-degree general disjunctions. Using general disjunctions to solve MIPs involves adding new inequality constraints to the LP sub-problems (i.e. child nodes) created from the current node. Figure 3 shows the child nodes of a typical general disjunction.

Figure 3: Child nodes of a general disjunction.

In the disjunction inequalities in Figure 3, $x_1, x_2, \dots, x_n$ are variables from the original MIP model, $a_1, a_2, \dots, a_n$ are coefficients that are restricted to a value of 0, 1, or -1, and k is the rounded down value of the disjunction equation using the LP-relaxation values of the variables from the parent node. An example of how to create the child nodes of a general disjunction is given in Section 5.4.

There are many different ways of forming 45-degree general disjunctions in MIPs. The algorithms in this thesis focus only on disjunctions that are created to be as parallel or as perpendicular as possible to an active constraint. In addition to being effective, this is more efficient because it limits the set of potential disjunctions that must be considered. The different heuristics used for choosing the best active constraint for branching are highlighted in Section 5.3. Heuristics governing the different ways of creating 45-degree general disjunctions that are as parallel or as perpendicular as possible to an active constraint are discussed in Section 5.4.

$$\begin{aligned} \text{MIP:} \quad & \text{Min } c_1x + c_2y \\ & x, y \in S \\ \\ \text{S:} \quad & A_1x + A_2y \{ \geq, \leq, = \} b \\ & x \in \mathbb{R}^d, y \in \mathbb{Z}^{n-d} \\ & l_1 \leq x \leq u_1, l_2 \leq y \leq u_2 \end{aligned}$$

Figure 4: Approximately parallel 45-degree general disjunction.

Figure 4 graphically illustrates a 45-degree general disjunction that is approximately parallel to an active constraint. The bold dot at the bottom right

corner of the feasible region in Figure 4 represents the initial LP-relaxation optimum, while the bold red line highlights the *branching active constraint*, i.e. the active constraint chosen for forming the approximately parallel or perpendicular general disjunction, and the two oblique dotted lines represent the approximately parallel 45-degree general disjunction.

Since general disjunctions normally branch on more than one variable at a time, they tend to force a greater number of variables to change and hence should lead to a MIP-feasible solution faster and in fewer nodes than traditional single-variable disjunctions. Using only coefficients of -1, 0, or +1 in a 45-degree disjunction makes sure that the disjunction equation passes through a set of integer-valued lattice points. By combining such disjunctions with single-variable disjunctions as mentioned in Section 5.1, the LP-feasible region can be shaped to create more MIP-feasible corner points.

5.3. Choosing an Active Constraint for Branching

Acquiring the first feasible solution as quickly as possible requires fast, efficient heuristics that target the right aspects of a MIP model. A 45-degree general disjunction already has the advantage of branching on multiple variables in the same disjunction while precluding the possibility of having MIP-feasible points between the branching inequalities. However, this might not be enough to guarantee fast MIP-feasibility, as other aspects of the algorithm such as where, how, and when general disjunctions should be used are all key decision points in the algorithm. This sub-section will generally focus on choosing the active constraint for general branching. Other aspects of constructing the disjunction

such as (i) choosing whether perpendicular or parallel to the branching constraint, and (ii) setting the coefficients, are discussed in Section 5.4.

Patel and Chinneck [2007] showed the impact of active constraints on achieving MIP-feasibility quickly. The authors discussed heuristics that involved the use of active constraints for deciding on the best branching variable. Such heuristics were experimentally proven to be very successful in finding the first MIP-feasible solution fast. Instead of considering the impact of the candidate variables on the objective function, they focused on candidate variables that have the greatest impact on the constraints that are active at the LP relaxation optimum. Similarly, the algorithms developed in this thesis identify the active constraint that has the most impact in the parent node and then base the general disjunction on that active constraint. "Impact" in the context of this thesis means changing the values of many candidate variables in order to reach a MIP-feasible solution. An active constraint that affects a large number of candidate variables is likely to have a great impact on achieving MIP-feasibility quickly. Using different criteria, the active constraints in the parent node are ranked. The highest-ranked active constraint is then used to create a "45-degree" general disjunction that is as parallel or as perpendicular as possible to that active constraint.

The set of active constraints includes all equality constraints and all inequality constraints that are tight at the current LP relaxation point [Patel and Chinneck, 2007]. By choosing the active constraint with the highest impact and creating a general disjunction that is approximately parallel or perpendicular to that constraint, there is a higher chance that one of the child nodes will lead to an

MIP-feasible solution. The different active-constraint ranking schemes use properties such as the total number of candidate variables in the active constraint, the sum of (integer/ candidate) variable absolute coefficients, and the feasibility gap sum to estimate the impact of the active constraints on the MIP solution.

The *candVarSum* measure used in some of the ranking schemes simply counts the total number of candidate variables in an active constraint. Similarly, the *intVarSum* measure counts the total number of integer variables present in an active constraint. The *sumCandCoeffs* measure is used in some schemes to hold the sum of candidate-variable absolute coefficients in an active constraint. Likewise, the *sumIntCoeffs* measure is used to hold the sum of integer-variable absolute coefficients in an active constraint. Some of the ranking schemes also use the *feasGapSum* measure, which represents the feasibility gap sum of an active constraint, which is the cumulative sum of the difference between the current value of each candidate variable and the closest integer value, for all candidate variables in the active constraint. Note that *candVarSum* and *feasGapSum* are dynamic measures as their values could change for the same constraint as the MIP solution moves from one LP optimum to another LP optimum. Since the number of candidate variables is constantly changing, the feasibility gap sum is also changing, which consequently changes the values of these measures. Table 2 summarizes the various algorithms used to choose an active constraint for branching. All branching-constraint selection algorithms rank

the best constraints using a specific criterion, and then break the tie between equally-ranked active constraints using another criterion.

Table 2: Summary of branching-constraint selection algorithms.

	Algorithm Description
1	<i>candVarSum_feasGapSum</i> Choose the active constraint with the largest number of candidate variables. Break ties between equally-ranked constraints using the highest feasibility-gap sum.
2	<i>intVarSum_sumIntCoeffs_largestIntCoeffNorm</i> Choose the active constraint with the largest number of integer variables. Break ties between equally-ranked constraints using the highest sum of integer-variable absolute coefficients, normalized by the largest absolute integer-variable coefficient.
3	<i>sumCandCoeffs_feasGapSum</i> Choose the active constraint with the highest sum of candidate-variable absolute coefficients (unnormalized). Break ties between equally-ranked constraints using the highest feasibility-gap sum.
4	<i>candVarSum_sumCandCoeffs_largestCandCoeffNorm</i> Choose the active constraint with the largest number of candidate variables. Break ties between equally-ranked constraints using the highest sum of candidate-variable absolute coefficients, normalized by the largest absolute candidate-variable coefficient.
5	<i>sumCandCoeffs_largestCandCoeffNorm_feasGapSum</i> Choose the active constraint with the highest sum of candidate-variable absolute coefficients, normalized by the largest absolute candidate-variable coefficient. Break ties between equally-ranked constraints using the highest feasibility gap sum.
6	<i>sumCandCoeffs_candVarSumNorm_feasGapSum</i> Choose the active constraint with the highest sum of candidate-variable absolute coefficients, normalized by the sum of candidate variables in the constraint. Break ties between equally-ranked constraints using the highest feasibility gap sum.

As can be seen in Table 2, the second, fourth, fifth, and sixth algorithms require some sort of normalization. The sum of absolute variable coefficients in these algorithms is normalized by the largest absolute variable coefficient or by the number of variables in the constraint. Normalization is especially required when the weights (i.e. coefficients or number of constraint variables) of the active

constraints in a MIP model are not evenly distributed. As a result, normalization is used to adjust the relative weight of each active constraint. Algorithms 3 and 6 can be used to compare the performance of an unnormalized algorithm versus a normalized version of the same algorithm. The various algorithms are used to determine the impact of an active constraint on the MIP solution using the sum of integer variables, the sum of candidate variables, the sum of absolute integer-variable coefficients, the sum of absolute candidate-variable coefficients, and/or the integer-feasibility gap sum.

An active constraint containing the largest number of integer or candidate variables is likely to have a great influence on the values of the candidate variables. By creating a 45-degree disjunction parallel or perpendicular to such a constraint, a large number of candidate variables will be forced to change to satisfy the added constraint, hence the chance of hitting a MIP-feasible solution increases. Similarly, an active constraint with the highest sum of integer-variable or candidate-variable absolute coefficients is likely to have the greatest impact on the MIP solution. Branching parallel or perpendicular to such constraints can also be very effective in finding the first MIP-feasible solution quickly. On a similar note, an active constraint with the highest feasibility gap sum is a constraint that has more influence on the feasibility of the MIP solution.

Input: MIP.

0. Initialize: $candVarSum \leftarrow 0$; $feasGapSum \leftarrow 0$; $highestCandVarSum \leftarrow 0$;
 $highestFeasGapSum \leftarrow 0$; $branchingConstraint \leftarrow 0$;

1. For each active constraint, c_i , in the LP relaxation solution:

1.1 For each candidate variable, x_i , in c_i :

1.1.1 $candVarSum \leftarrow candVarSum + 1$

1.1.2 If $(x_i - \lfloor x_i \rfloor) < 0.5$ then

1.1.2.1 $feasGapSum \leftarrow feasGapSum + (x_i - \lfloor x_i \rfloor)$

1.1.3 Else: $feasGapSum \leftarrow feasGapSum + (\lfloor x_i \rfloor - x_i)$

1.2 If $candVarSum > highestCandVarSum$ then

1.2.1 $highestCandVarSum \leftarrow candVarSum$

1.2.2 $highestFeasGapSum \leftarrow feasGapSum$

1.2.3 $branchingConstraint \leftarrow c_i$

1.3 Else if $(candVarSum = highestCandVarSum)$ and $(feasGapSum > highestFeasGapSum)$ then

1.3.1 $highestFeasGapSum \leftarrow feasGapSum$

1.3.2 $branchingConstraint \leftarrow c_i$

2. Create a parallel or perpendicular disjunction to the active constraint
 $branchingConstraint$.

Algorithm 5: Pseudocode for the $candVarSum_feasGapSum$ branching-constraint selection algorithm.

Pseudocodes for the different branching-constraint selection algorithms listed in

Table 2 are provided in Algorithm 5 through Algorithm 10.

Input: MIP.

0. Initialize: $intVarSum \leftarrow 0$; $sumIntCoeffs \leftarrow 0$; $highestIntVarSum \leftarrow 0$;
 $largestIntCoeff \leftarrow 0$; $highestIntCoeffs \leftarrow 0$; $branchingConstraint \leftarrow 0$;

1. For each active constraint, c_i , in the LP relaxation solution:

1.1 For each integer variable, x_i , in c_i :

1.1.1 $intVarSum \leftarrow intVarSum + 1$

1.1.2 $sumIntCoeffs \leftarrow sumIntCoeffs + |\text{coefficient of } x_i|$

1.1.3 If $|\text{coefficient of } x_i| > largestIntCoeff$ then

1.1.3.1 $largestIntCoeff \leftarrow |\text{coefficient of } x_i|$

1.2 If $intVarSum > highestIntVarSum$ then

1.2.1 $highestIntVarSum \leftarrow intVarSum$

1.2.2 $highestSumIntCoeffs \leftarrow (sumIntCoeffs / largestIntCoeff)$

1.2.3 $branchingConstraint \leftarrow c_i$

1.3 Else if $(intVarSum = highestIntVarSum)$ and $((sumIntCoeffs / largestIntCoeff) > highestSumIntCoeffs)$ then

1.3.1 $highestSumIntCoeffs \leftarrow (sumIntCoeffs / largestIntCoeff)$

1.3.2 $branchingConstraint \leftarrow c_i$

2. Create a parallel or perpendicular disjunction to the active constraint $branchingConstraint$.

Algorithm 6: Pseudocode for the $intVarSum_sumIntCoeffs_largestIntCoeffNorm$ branching-constraint selection algorithm.

Input: MIP.

0. Initialize: $sumCandCoeffs \leftarrow 0$; $feasGapSum \leftarrow 0$; $highestSumCandCoeffs \leftarrow 0$;
 $highestFeasGapSum \leftarrow 0$; $branchingConstraint \leftarrow 0$;

1. For each active constraint, c_i , in the LP relaxation solution:

1.1 For each candidate variable, x_i , in c_i :

1.1.1 $sumCandCoeffs \leftarrow sumCandCoeffs + |\text{coefficient of } x_i|$

1.1.2 If $(x_i - \lfloor x_i \rfloor) < 0.5$ then

1.1.2.1 $feasGapSum \leftarrow feasGapSum + (x_i - \lfloor x_i \rfloor)$

1.1.3 Else: $feasGapSum \leftarrow feasGapSum + (\lfloor x_i \rfloor - x_i)$

1.2 If $sumCandCoeffs > highestSumCandCoeffs$ then

1.2.1 $highestSumCandCoeffs \leftarrow sumCandCoeffs$

1.2.2 $highestFeasGapSum \leftarrow feasGapSum$

1.2.3 $branchingConstraint \leftarrow c_i$

1.3 Else if $(sumCandCoeffs = highestSumCandCoeffs)$ and $(feasGapSum > highestFeasGapSum)$ then

1.3.1 $highestFeasGapSum \leftarrow feasGapSum$

1.3.2 $branchingConstraint \leftarrow c_i$

2. Create a parallel or perpendicular disjunction to the active constraint $branchingConstraint$.

Algorithm 7: Pseudocode for the $sumCandCoeffs_feasGapSum$ branching-constraint selection algorithm.

Input: MIP.

0. Initialize: $candVarSum \leftarrow 0$; $sumCandCoeffs \leftarrow 0$; $highestCandVarSum \leftarrow 0$;
 $largestCandCoeff \leftarrow 0$; $highestCandCoeffs \leftarrow 0$; $branchingConstraint \leftarrow 0$;
1. For each active constraint, c_i , in the LP relaxation solution:
- 1.1 For each candidate variable, x_i , in c_i :
 - 1.1.1 $candVarSum \leftarrow candVarSum + 1$
 - 1.1.2 $sumCandCoeffs \leftarrow sumCandCoeffs + |\text{coefficient of } x_i|$
 - 1.1.3 If $|\text{coefficient of } x_i| > largestCandCoeff$ then
 - 1.1.3.1 $largestCandCoeff \leftarrow |\text{coefficient of } x_i|$
 - 1.2 If $candVarSum > highestCandVarSum$ then
 - 1.2.1 $highestCandVarSum \leftarrow candVarSum$
 - 1.2.2 $highestSumCandCoeffs \leftarrow (sumCandCoeffs / largestCandCoeff)$
 - 1.2.3 $branchingConstraint \leftarrow c_i$
 - 1.3 Else if $(candVarSum = highestCandVarSum)$ and $((sumCandCoeffs / largestCandCoeff) > highestSumCandCoeffs)$ then
 - 1.3.1 $highestSumCandCoeffs \leftarrow (sumCandCoeffs / largestCandCoeff)$
 - 1.3.2 $branchingConstraint \leftarrow c_i$
2. Create a parallel or perpendicular disjunction to the active constraint $branchingConstraint$.

Algorithm 9: Pseudocode for the *candVarSum_sumCandCoeffs_largestCandCoeffNorm* branching-constraint selection algorithm.

Input: MIP.

0. Initialize: $sumCandCoeffs \leftarrow 0$; $feasGapSum \leftarrow 0$; $largestCandCoeff \leftarrow 0$;
 $highestSumCandCoeffs \leftarrow 0$; $highestFeasGapSum \leftarrow 0$;
 $branchingConstraint \leftarrow 0$;
1. For each active constraint, c_i , in the LP relaxation solution:
- 1.1 For each candidate variable, x_i , in c_i :
 - 1.1.1 $sumCandCoeffs \leftarrow sumCandCoeffs + |\text{coefficient of } x_i|$
 - 1.1.2 If $|\text{coefficient of } x_i| > largestCandCoeff$ then
 - 1.1.2.1 $largestCandCoeff \leftarrow |\text{coefficient of } x_i|$
 - 1.1.3 If $(x_i - \lfloor x_i \rfloor) < 0.5$ then
 - 1.1.3.1 $feasGapSum \leftarrow feasGapSum + (x_i - \lfloor x_i \rfloor)$
 - 1.1.4 Else: $feasGapSum \leftarrow feasGapSum + (\lfloor x_i \rfloor - x_i)$
 - 1.2 If $(sumCandCoeffs / largestCandCoeff) > highestSumCandCoeffs$ then
 - 1.2.1 $highestSumCandCoeffs \leftarrow (sumCandCoeffs / largestCandCoeff)$
 - 1.2.2 $highestFeasGapSum \leftarrow feasGapSum$
 - 1.2.3 $branchingConstraint \leftarrow c_i$
 - 1.3 Else if $((sumCandCoeffs / largestCandCoeff) = highestSumCandCoeffs)$ and $(feasGapSum > highestFeasGapSum)$ then
 - 1.3.1 $highestFeasGapSum \leftarrow feasGapSum$
 - 1.3.2 $branchingConstraint \leftarrow c_i$
2. Create a parallel or perpendicular disjunction to the active constraint $branchingConstraint$.

Algorithm 8: Pseudocode for the *sumCandCoeffs_largestCandCoeffNorm_feasGapSum* branching-constraint selection algorithm.

Input: MIP.

0. Initialize: $sumCandCoeffs \leftarrow 0$; $candVarSum \leftarrow 0$; $feasGapSum \leftarrow 0$;
 $highestSumCandCoeffs \leftarrow 0$; $highestFeasGapSum \leftarrow 0$;
 $branchingConstraint \leftarrow 0$;

1. For each active constraint, c_i , in the LP relaxation solution:

1.1 For each candidate variable, x_i , in c_i :

1.1.1 $sumCandCoeffs \leftarrow sumCandCoeffs + |\text{coefficient of } x_i|$

1.1.2 If $(x_i - \lfloor x_i \rfloor) < 0.5$ then

1.1.2.1 $feasGapSum \leftarrow feasGapSum + (x_i - \lfloor x_i \rfloor)$

1.1.3 Else: $feasGapSum \leftarrow feasGapSum + (\lfloor x_i \rfloor - x_i)$

1.1.4 $candVarSum \leftarrow candVarSum + 1$

1.2 If $(sumCandCoeffs / candVarSum) > highestSumCandCoeffs$ then

1.2.1 $highestSumCandCoeffs \leftarrow (sumCandCoeffs / candVarSum)$

1.2.2 $highestFeasGapSum \leftarrow feasGapSum$

1.2.3 $branchingConstraint \leftarrow c_i$

1.3 Else if $((sumCandCoeffs / candVarSum) = highestSumCandCoeffs)$ and
 $(feasGapSum > highestFeasGapSum)$ then

1.3.1 $highestFeasGapSum \leftarrow feasGapSum$

1.3.2 $branchingConstraint \leftarrow c_i$

2. Create a parallel or perpendicular disjunction to the active constraint
 $branchingConstraint$.

Algorithm 10: Pseudocode for the $sumCandCoeffs_candVarSumNorm_feasGapSum$ branching-constraint selection algorithm.

Experiments conducted in Section 7 compare the different branching-constraint selection algorithms. Such experiments help determine the factors and parameters that most influence the efficient creation of a 45-degree general disjunction that is approximately parallel or perpendicular to a chosen active constraint. Selecting the right branching constraint determines the set of candidate variables that will be impacted the most by the resulting 45-degree general disjunction.

5.4. Forming a 45-Degree General Disjunction Based on an Active Constraint

Once an active constraint is chosen for branching, the next step is to create a 45-degree general disjunction that is as parallel or as perpendicular as possible to that active constraint. Whether the 45-degree disjunction is most parallel to, or most perpendicular to, the branching constraint, depends on the constraint type. For inequality branching constraints, there is the choice of creating a parallel or perpendicular disjunction. However, branching as parallel as possible to an equality constraint is usually ineffective, especially when the parallel branch lies exactly on the equality constraint. Figure 5 illustrates such a scenario in a 3-D MIP problem, where the down branch of the parallel disjunction lies exactly on the branching equality constraint, and the up branch of the disjunction is infeasible. The feasible region shown in Figure 5 represents the " y_1 - y_2 " subspace of the branching constraint in the 3-D MIP model. The scenario shown arises when the branching active constraint is also a 45-degree equality constraint, in which case all the coefficients of the 45-degree disjunction equation are the same as the branching-constraint equation.

Figure 5: Branching parallel to a 45-degree equality constraint.

In order to avoid scenarios similar to Figure 5, the algorithms developed in this thesis use approximately perpendicular general disjunctions for equality branching constraints. On the other hand, approximately parallel general disjunctions work well with inequality constraints as one side of the disjunction leads to tiny feasible region that should solve quickly and the other side has a nice edge that passes through the integer lattice points, as seen in Figure 4. As a result, approximately parallel general disjunctions are used with inequality branching constraints; thus the type of 45-degree disjunction created will depend on the type of branching constraint.

Forming an approximately parallel 45-degree disjunction basically requires the signs of the variable coefficients in the disjunction equation to be the same as the signs of the coefficients of the variables in the branching-constraint equation. For example, if the branching constraint is as follows:

$$2x_1 - 7x_2 + 15x_3 \leq 30,$$

where x_1 , x_2 , and x_3 are integer variables with non-negativity variable bounds and LP-relaxation values of 4.6, 3.2, and 2.88 respectively, then the approximately parallel 45-degree disjunction will be as follows:

$$\text{Down branch: } x_1 - x_2 + x_3 \leq 4,$$

$$\text{Up branch: } x_1 - x_2 + x_3 \geq 5,$$

Where the right-hand side values of the up and down branches are calculated using the LP-relaxation values of x_1 , x_2 , and x_3 as follows:

$$\text{RHS of down branch: } \lfloor 4.6 - 3.2 + 2.88 \rfloor = \lfloor 4.28 \rfloor = 4$$

$$\text{RHS of up branch: } \lfloor 4.6 - 3.2 + 2.88 \rfloor + 1 = \lfloor 4.28 \rfloor + 1 = 5$$

The general algorithm for forming a 45-degree parallel disjunction is shown in Algorithm 11.

Input: MIP, branching constraint C .

0. Initialize: $rhs \leftarrow 0$;
1. For all variables, x_i , in C .
 - 1.1 If the coefficient of variable x_i in C is positive, then
 - 1.1.1 Set the coefficient of x_i in the branching equation to 1
 - 1.1.2 $rhs \leftarrow rhs + x_i$
 - 1.2 Else if the coefficient of variable x_i in C is negative, then
 - 1.2.1 Set the coefficient of x_i in the branching equation to -1
 - 1.2.1 $rhs \leftarrow rhs - x_i$
2. Create the down branch of the parallel disjunction with a right-hand side value of $\lfloor rhs \rfloor$
3. Create the up branch of the parallel disjunction with a right-hand side value of $\lfloor rhs \rfloor + 1$

Algorithm 11: Creating a 45-degree parallel disjunction.

5.4.1. Creating a Perpendicular General Disjunction

Creating a 45-degree general disjunction that is approximately parallel to a branching constraint is relatively easy, but creating an approximately

perpendicular 45-degree disjunction is not as natural. Creating a 45-degree general disjunction that is nearly perpendicular to a branching constraint is essentially the same as creating a general disjunction that is exactly perpendicular to the nearly parallel 45-degree general disjunction. In order to create an approximately perpendicular general disjunction, one must first find the normal vector of the nearly parallel plane. For example, considering the branching constraint provided in the previous example, the normal vector of the nearly parallel 45-degree plane is as follows:

$$v_{\text{para}} = (1, -1, 1),$$

which is the set of non-zero variable coefficients of the nearly parallel 45-degree disjunction equation. Given this normal vector, v_{para} , one can now find an approximately perpendicular general disjunction by finding a normal vector, v_{perp} , which makes the dot product of v_{para} and v_{perp} equal to zero. One possible normal vector perpendicular to the above parallel plane could be the following:

$$v_{\text{perp}} = (-1, -1, 0),$$

which is a valid perpendicular normal vector as the dot product of v_{para} and v_{perp} is equal to zero, as shown below:

$$\begin{aligned} v_{\text{para}} \cdot v_{\text{perp}} &= (1, -1, 1) \cdot (-1, -1, 0) \\ &= 0. \end{aligned}$$

In order to create a valid perpendicular normal vector, two decisions must be made: (i) if there exists an odd number of nonzeros in v_{para} , then which coefficient to set to zero, and (ii) how to set the signs on the remaining nonzeros in the perpendicular normal vector. When the number of nonzero variable

coefficients in v_{para} is odd, at least one of the coefficients corresponding to a nonzero variable coefficient in v_{perp} must be set to zero. Given that the objective is to create a 45-degree disjunction that is as perpendicular as possible to the branching constraint, the number of coefficients required to be set to zero should be minimized. This means that only one variable coefficient should be set to zero when the number of nonzero coefficients in the parallel disjunction equation is odd. When such a situation arises, a variable selection scheme is used to decide on the variable whose coefficient will be set to zero.

Different variable selection schemes aim at choosing the variable which has the least impact on the MIP solution. For instance, choosing to set the coefficient of a continuous variable to zero will tend to have less impact on the MIP solution than choosing a candidate variable. This is because the main purpose of a general disjunction is to force as many candidate variables as possible to change.

Another possible variable selection scheme could choose the variable with the smallest absolute coefficient in the branching equation. Following the same logic, setting the coefficient of the variable with the smallest absolute coefficient in the branching equation to zero would have less impact on the MIP solution as opposed to a variable with a large coefficient. Whether that variable with the smallest absolute coefficient is a candidate variable or not might also have an impact on the performance of the algorithm.

Choosing the candidate variable with the smallest integer infeasibility is yet another variable selection scheme. A candidate variable with a large integer

infeasibility will tend to be influenced more by the general disjunction than a candidate variable that is close to integrality (a.k.a quasi-integer variable). Benichou et al. [1971] noted that branching on quasi-integer variables should be delayed as much as possible as it is expected that, in the nearest integer solution, these variables will take on the nearest integer value. Consequently, setting the coefficient of a quasi-integer variable to zero should have the least impact on the MIP solution.

Another variable selection algorithm is based on selecting the variable that is present in the least number of active constraints. Once again, setting the coefficient of such a variable to zero in the perpendicular disjunction will have less impact on the MIP solution than choosing a variable that is present in a large number of active constraints. Variables that are present in many active constraints are more likely to propagate the impact of the general disjunction as they are forced to change in order satisfy the various constraints they appear in.

Once one has decided on which variable coefficient to set to zero, the next step is to decide on the ± 1 settings of variable coefficients that will make up the normal vector of the perpendicular plane. As the number of variables present in the branching constraint increases, the number of possible combinations for the normal vector of the perpendicular plane increases rapidly. We focus on two of the many possible combinations for the perpendicular normal vector. One way of creating the normal vector for the perpendicular plane is by switching the sign of every other non-zero coefficient in the parallel normal vector. Switching sign on odd counter or on even counter are just two of the many different ways of

creating a perpendicular normal vector. The algorithm for creating a 45-degree perpendicular disjunction by switching sign on odd counter is given below in Algorithm 12.

```

Input: MIP, branching constraint C.
0. Initialize:  $rhs \leftarrow 0$ ;  $counter \leftarrow 0$ ;
1. If the number of non-zero coefficients in C is odd, then
    1.1 Use a variable selection scheme to select a variable  $v$ 
2. For all variables,  $x_i$ , in C:
    2.1 If  $x_i$  is  $v$ , then
        2.1.1 Set the coefficient of  $v$  in the branching equation to 0
        2.1.2 Go to Step 2 (i.e. next variable in C)
    2.2 If  $counter$  is even, then
        2.2.1 If the coefficient of variable  $x_i$  in C is positive, then
            2.2.1.1 Set the coefficient of  $x_i$  in the branching equation to 1
            2.2.1.2  $rhs \leftarrow rhs + x_i$ 
        2.2.2 Else if the coefficient of variable  $x_i$  in C is negative, then
            2.2.2.1 Set the coefficient of  $x_i$  in the branching equation to -1
            2.2.2.1  $rhs \leftarrow rhs - x_i$ 
    2.3 Else if  $counter$  is odd, then
        2.3.1 If the coefficient of variable  $x_i$  in C is positive, then
            2.3.1.1 Set the coefficient of  $x_i$  in the branching equation to -1
            2.3.1.2  $rhs \leftarrow rhs - x_i$ 
        2.3.2 Else if the coefficient of variable  $x_i$  in C is negative, then
            2.3.2.1 Set the coefficient of  $x_i$  in the branching equation to 1
            2.3.2.1  $rhs \leftarrow rhs + x_i$ 
    2.4  $counter \leftarrow counter + 1$ 
3. Create the down branch of the perpendicular disjunction with a right-hand side value of  $\lfloor rhs \rfloor$ 
4. Create the up branch of the perpendicular disjunction with a right-hand side value of  $\lfloor rhs \rfloor + 1$ 

```

Algorithm 12: Creating a 45-degree perpendicular disjunction.

Experiments testing the different variable selection schemes used in creating the 45-degree perpendicular disjunction are presented in Section 7. Additional experiments testing the effects of switching sign on odd counter or on even counter are also provided in Section 7.

5.5. Branching Direction of General Disjunctions

Deciding on the branching direction of disjunctions in MIPs is an ongoing research topic. With every disjunction, whether axis-parallel or general, there are two possible branching choices: branching in the direction of the up node or branching in the direction of the down node. Deciding which branch to explore first depends on the branching direction selection scheme used. Different branching direction selection schemes may consider different parameters with the intention of choosing the most promising child node in the B&B tree.

Previous research on branching direction focused on axis-parallel disjunctions, where the choice of branching direction may depend on the choice of branching variable [Pryor and Chinneck, 2011] [Achterberg et al., 2005] [Meyer et al., 2003]. Since general disjunctions usually involve more than one candidate variable, the choice of branching direction should aim at forcing as many candidate variables as possible to change. Pryor and Chinneck [2011] developed new methods based on estimating the probability of satisfying a constraint at the child node given a branching variable/direction pair. Surprisingly, they discovered that feasibility is usually achieved more quickly when the branching variable direction pair with the smallest probability of satisfying the constraint is selected. A similar principle is applied here.

The best branching direction for a general disjunction depends on the coefficients of the disjunction variables as well as the RHS values of the branches. Since the 45-degree general disjunctions developed in this thesis have only $\{-1, 0, 1\}$ coefficients that are based on the coefficients of the branching

active constraint, choosing a branching direction mainly boils down to choosing the direction and the RHS value of the disjunction function. The following branching direction selection schemes are considered in this thesis: (i) branching to increase or decrease the value of the disjunction function, (ii) branching to the side of the disjunction that is closer to or farther from the current LP-relaxation optimum point, and (iii) branching in the satisfying direction of the branching active constraint inequality or the opposite direction. The third branching direction selection scheme is only applicable to parallel general disjunctions as it requires the use of an inequality branching constraint which does not apply for perpendicular general disjunctions. More details on the third branching direction selection scheme are provided in Section 5.5.1.

Branching to increase or decrease the value of the disjunction function is analogous to the classical branching in the up or down direction schemes. Using the example presented in Section 5.4, where the up and down branches are as follows:

$$\text{Down branch: } x_1 - x_2 + x_3 \leq 4,$$

$$\text{Up branch: } x_1 - x_2 + x_3 \geq 5,$$

where x_1 , x_2 , and x_3 are integer variables with non-negativity bounds and LP-relaxation values of 4.6, 3.2, and 2.88 respectively.

Branching to increase the value of the disjunction function in the above example is taking the up branch, whereas branching to decrease the value of the disjunction function is taking the down branch. Unlike other branching direction selection schemes, algorithms using this scheme will constantly take the up

branch every time, or the down branch every time, regardless of any other factors. Branching to increase or decrease the value of the disjunction function are branching direction selection schemes that are valid for algorithms using either 45-degree parallel or perpendicular general disjunctions.

While branching to increase or decrease the value of the disjunction function does not consider any other factors in the selection process, branching closer to or farther from the LP-relaxation optimum point takes into account the location of the LP-relaxation optimum point. To determine which branch is closer to or farther from the LP-relaxation optimum point, first calculate the value of the disjunction function at the LP-relaxation point. In the example shown in section 5.4, the value of the disjunction function at the LP-relaxation point is calculated as follows:

$$x_1 - x_2 + x_3 = 4.6 - 3.2 + 2.88 = 4.28,$$

which lies between the up and down branches of the general disjunction. Since 4.28 is closer to 4 than it is to 5, the down direction is closer to the LP-relaxation optimum point. For approximately parallel 45-degree disjunctions, branching closer to the LP-relaxation optimum point is not necessarily the same as branching towards the feasible region, as will be seen in Section 5.5.1.

Branching farther from the LP-relaxation optimum point is an interesting tactic as it forces many candidate variables to change at once in order to satisfy the added disjunction constraint. This increases the chances of reaching a MIP-feasible solution. These schemes are applicable to algorithms involving parallel or perpendicular 45-degree general disjunctions. The general algorithms for

branching closer to or farther from the LP-relaxation optimum point are provided in Algorithm 13 and Algorithm 14, respectively. Experiments revealing the effects of branching to force change by branching farther from the LP-relaxation optimum point are presented in Section 7.

Input: MIP, rhs of disjunction function at the LP-relaxation optimum.
 1. If $(rhs - \lfloor rhs \rfloor) < 0.5$, then
 1.1 Branch in the down direction with a right-hand side value of $\lfloor rhs \rfloor$
 2. Else branch in the up direction with a right-hand side value of $\lfloor rhs \rfloor + 1$

Algorithm 13: Branching closer to the LP-relaxation optimum point.

Input: MIP, rhs of disjunction function at the LP-relaxation optimum.
 1. If $(rhs - \lfloor rhs \rfloor) < 0.5$, then
 1.1 Branch in the up direction with a right-hand side value of $\lfloor rhs \rfloor + 1$
 2. Else branch in the down direction with a right-hand side value of $\lfloor rhs \rfloor$

Algorithm 14: Branching farther from the LP-relaxation optimum point.

5.5.1. More on Branching Direction Selection Schemes for Parallel General

Disjunctions

Approximately parallel 45-degree general disjunctions are only applied if the branching constraint is an inequality constraint, otherwise perpendicular general disjunctions are used. Since the inequality branching constraint on which the general disjunction is based defines the direction of the feasible region, one has the choice of branching either in the satisfying direction of the branching constraint or in the opposite direction. Branching in the satisfying direction of the branching constraint or in the opposite direction are branching direction selection schemes that are exclusive to parallel 45-degree general disjunctions.

Using Pryor and Chinneck's [2011] ideas on branching to force change, we observe that branching in the satisfying direction of the branching constraint further constricts the feasible region and thus a number of candidate variables

will be forced to change in order to satisfy the added disjunction constraint. Furthermore, branching in the satisfying direction of the branching constraint will usually make sure that the resulting LP sub-problem at the child node is feasible, provided that the original MIP model has a feasible solution.

One could also branch in the violating direction of the branching constraint. However, branching in the violating direction of the branching constraint is not useful as it most often leads to an infeasible child node. Figure 6 provides a graphical representation of the following two-dimensional MIP model:

$$\text{Max } Z = y_1 + 10y_2$$

Subject to

$$-25y_1 + 20y_2 \leq 30$$

$$y_1 + 0.4y_2 \leq 4,$$

where y_1 and y_2 are integer variables with variable bounds $y_1, y_2 \geq 0$.

Figure 6: Branching in the satisfying or the violating direction for the branching constraint.

The bold lines in Figure 6 represent the constraints of the MIP model, and the dotted line represents the objective function at the LP-optimum. The LP-relaxation values of y_1 and y_2 at the LP optimum are 1.5 and 3.4 respectively, which are used in creating the 45-degree general disjunction that is approximately parallel to the indicated branching constraint. The added constraints at the child nodes are the following:

$$\text{Down branch: } -y_1 + y_2 \leq 1$$

$$\text{Up branch: } -y_1 + y_2 \geq 2,$$

where the RHS values of the disjunction equations are calculated using the LP-relaxation values at the LP-relaxation optimum point. Branching in the satisfying direction of the branching constraint, or in other words towards the feasible region, is achieved in the above example by taking the down branch. On the other hand, taking the up branch is branching in the violating direction of the branching constraint.

Branching in the satisfying direction has the advantage of forcing many candidate variable to change at once, but in some situations it may be ineffective. When the branching constraint is a 45-degree inequality constraint itself (such as a multiple-choice constraint), one of the created branches will lie exactly on the branching constraint. This is similar to the scenario presented in Figure 5 where the branching constraint is a 45-degree equality constraint and creates an ineffective general disjunction that is parallel to the branching constraint. In order to avoid such scenarios, a parallel 45-degree general disjunction that is offset by one unit can be created instead. This further constricts the feasible region, thus

speeding the process of attaining a MIP-feasible solution. Experiments in Section 7 compare the speed of the MIP solutions for the various branching-direction selection schemes with or without offsetting the parallel general disjunction by one unit.

6. Experimental Setup

Experimental conditions including software, hardware, solver parameter settings, test sets, and comparison metrics are discussed in this section.

6.1. Software

The well-known commercial solver Cplex 12.1 [Ilog 2009a, 2009b, 2009c] is used in all the experiments reported in Sections 7 and 8. Cplex is used as the state-of-the-art commercial MIP solver with which the algorithms developed in this thesis are compared. Cplex is also used as the basic MIP solver upon which new algorithms are developed.

6.1.1 Programming Language and Compiler

The algorithms developed in this thesis are all written in the programming language C, which is one of the languages for which Cplex provides an interface to the solver's parameters and functions [Ilog 2009a, 2009b]. Microsoft Visual Studio 2008 [Microsoft 2007] was the development environment used for integrating Cplex's DLLs (Dynamic Link Libraries) and compiling the algorithms written in C.

6.1.2 Cplex Parameter Settings

All the Cplex parameters mentioned in the parameters' reference manual [Ilog 2009a] are set at their default values in all experiments, with the following exceptions (the corresponding Cplex parameter is provided in brackets):

- Number of MIP solutions to be found before stopping (*CPX_PARAM_INTSOLLIM*): This parameter is set to 1 as the objective of the algorithms is to attain only the first MIP-feasible solution.

- Node selection strategy (*CPX_PARAM_NODESEL*) : The node selection strategy defines the rule for the next node to explore when backtracking. Depth-first node selection, which picks the most recently created node, is chosen.
- Time Limit (*CPX_PARAM_TILIM*): The maximum CPU time limit is set to two hours (7200 seconds) for experiments presented in Section 7, but is set to eight hours (28800 seconds) for the final experiments presented in Section 8.
- Default number of parallel threads invoked by any CPLEX parallel optimizer (*CPX_PARAM_THREADS*): all experiments run on single threads.
- MIP emphasis (*CPX_PARAM_MIPEMPHASIS*): Emphasis is on MIP-feasibility.

In addition to the above parameter settings, most of the node pre-processing and node heuristics in the underlying Cplex solver are turned off in all experiments by these additional parameter settings:

- Pre-solving (*CPX_PARAM_PREIND*): off.
- Aggregation (*CPX_PARAM_AGGIND*): off.
- Internal Node Heuristic (*CPX_PARAM_HEURFREQ*): off.
- Cut generation (*CPX_PARAM_CLIQUES*, *CPX_PARAM_COVERS*, *CPX_PARAM_CUTPASS*, *CPX_PARAM_CUTSFACTOR*, *CPX_PARAM_DISJCUTS*, *CPX_PARAM_FLOWCOVERS*, *CPX_PARAM_FLOWPATHS*, *CPX_PARAM_FRACCUTS*,

CPX_PARAM_GUBCOVERS, *CPX_PARAM_IMPLBD*,
CPX_PARAM_MCFCUTS, *CPX_PARAM_MIRCUTS*,
CPX_PARAM_ZEROHALFCUTS): off.

The *Baseline_Cplex* method, which represents the state-of-the-art algorithm for solving MIPs, is compared against the best algorithms developed on top of Cplex in Section 8. For a fair comparison, all of the above parameter settings are also applied to this method. Furthermore, since a few solver parameters such as *dynamic search* [Ilog 2009a] are switched off by default when a callback routine (see Section 6.1.3 for definition) is invoked, thus *Baseline_Cplex* is provided with dummy callback routines, which are simply empty callback routines.

MIP models are run until they attain a MIP-feasible solution or until they reach the specified time limit. Therefore the Solution time for a MIP model in an experiment is MIN {time to first MIP-feasible solution, time limit}, where the time limit is either two hours or eight hours as mentioned in the parameter settings.

6.1.3 Programming the New Algorithms

Given the Cplex C API (application programming interface) [Ilog 2009b], a C program was written to manage the interface between the Cplex solver and the new algorithms. Interfacing with the solver during the MIP solution can be accomplished through callback routines that are provided by Cplex.

At various points in the solution process, different callback routines defining additional user functionalities are called by the Cplex MIP solver. Most of the algorithms developed here use two main callback routines which are set by two Cplex functions (see the Cplex manual [Ilog 2009b]): *CPXsetbranchcallbackfunc* and *CPXsetnodecallbackfunc*. *CPXsetbranchcallbackfunc* sets the user-written

callback routine to be called after a branch has been selected but before the branch is carried out during MIP optimization. This callback routine can be used to create a general disjunction instead of the default axis-parallel disjunction, when required. *CPXsetnodecallbackfunc* is another function that sets the user-written callback routine to be called after Cplex has selected a node to explore, but before this exploration is carried out. This callback routine is used to determine the number of *backtracks*, i.e. the number of times the solver encounters an LP-infeasible node and thus explores a different node in the B&B tree. The number of backtracks is one type of performance metric that will be further discussed in Section 6.4.

6.2. Hardware

Three computers running Microsoft Windows XP are used for the experiments:

- Two separate Intel Core 2 CPUs running at 2.40GHz and with 3GB of RAM
- An Intel Core 2 Quad CPU running at 2.40GHz and with 8GB of RAM

Methods from the same experiment are run on the same computer in order to allow for a fair comparison; nonetheless, it is possible to compare experiments run on different computers as the specifications for each of the cores in all three computers are fairly similar.

6.3. Test Models

Different types of MIP models, 350 in total, representing a variety of real-world problems, are taken from MIPLIB2010 [Koch, T. et al. 2011] to form the test

sets for the experiments presented in Sections 7 and 8. Different experiments use different subsets of the MIP problems.

Experiments comparing the algorithms for creating an approximately 45-degree perpendicular general disjunction, as well as experiments comparing the different branching direction selection schemes for perpendicular disjunctions, both use a subset comprised of MIP problems that contain at least one equality constraint. The presence of equality constraints in a MIP model does not guarantee the use of approximately perpendicular general disjunctions in the MIP solution, however, it at least creates opportunities for using such disjunctions. Since all the algorithms in the experiments keep track of the number of parallel and perpendicular general disjunctions that were created during the solution process, one can then disregard those MIP problems that did not utilize any perpendicular general disjunctions in their solutions. It is also useful to focus on those problems that only use approximately perpendicular general disjunctions in their solutions, in other words, those problems that did not use any parallel general disjunctions in their solutions. The subset of models having at least one equality constraint is provided in Appendix A.1.

Similarly, experiments comparing the various branching direction selection schemes for parallel disjunctions use a subset of models consisting of MIP problems that contain at least one inequality constraint. The subsequent analysis can ignore those models that did not generate any approximately parallel general disjunctions or can focus on those models that only utilized parallel general disjunctions in their solutions. The subset of models having at least one

inequality constraint, which is used to study the branching direction selection schemes for approximately parallel general disjunctions, is provided in Appendix A.2.

Additional experiments on deciding when to perform a general disjunction and branching-constraint selection experiments use a subset of models that have various different properties. The subset of models used in these experiments are given in Appendix A.3.

The 350 MIP models taken from MIPLIB2010 are divided into 2 sets: *training set* and *testing set*. The training set consists of models that run in less than a hour using the *Baseline_Cplex* algorithm. 239 of the 350 models are solved in less than an hour using *Baseline_Cplex* (either to find the first feasible solution or to discover that the problem is MIP-infeasible). Out of those 239 models, different subsets of 100 models are chosen randomly to constitute the different sets provided in Appendices A.1, A.2, and A.3. The training set in Appendices A.1, A.2, and A.3 contains 127 distinct MIP models. Consequently, there are 112 easy models (i.e. models that are solved in less than an hour using *Baseline_Cplex*) left from the 239 models which become part of the testing set. These models are listed in Appendix A.4. In addition to these models, the 111 hard models remaining (i.e. models that are solved in more than an hour using *Baseline_Cplex*) are also part of the testing set that is used for experiments in Section 8 (see list of hard models in Appendix A.5).

6.4. Metrics

In order to compare the different algorithms against each other or against *Baseline_Cplex*, three main performance metrics are used: total CPU time, number of simplex iterations completed, and the number of solved nodes. The number of solved nodes performance measure mainly captures the average memory consumption of an algorithm and can sometimes be related to the total computation effort [Patel and Chinneck, 2007]. However, it is not always a good measure of speed as an algorithm that has many solved nodes may finish in less time than one with a smaller number of solved nodes. This is because solving the child of a node that was just solved allows the solver to use a hot start on the LP sub-problem to yield a solution in just a few more iterations; whereas solving a node somewhere else in the tree does not enable the use of a hot start and thus it takes longer to solve the LP sub-problem at that node.

The total CPU time required to solve a MIP model is the most important metric from the user's point of view, but it is difficult to measure in modern multi-core machines. There is no guarantee that the timestamp counters of the different cores on a single computer are synchronized, and the CPU speeds may be adjusted for power-saving reasons depending on the load on the cores and periodic tasks carried out by the operating system.

For these reasons, we instead rely on the number of simplex iterations as the main metric for computational effort. This metric is a good proxy for computational time provided there is no significant computational effort outside of the LP solutions. Since the algorithms developed here use axis-parallel

disjunctions by default and occasionally use general disjunctions when required, the non-simplex computation time is mainly used for creating and applying general disjunctions, hence non-simplex time heavily depends on the number of general disjunctions created. It also depends on the method used for choosing the branching constraint and for creating the approximately parallel and perpendicular general disjunctions. The average time spent in the callback routines (which is the time for setting up general disjunctions) for the final algorithms compared in Section 8 with respect to the total time spent executing the set of Models given in Appendix A.4 is approximately 0.3%. The maximum percentage of time spent in the callback routines for any model in the subset is 4%, which even then is negligible. For these reasons, the number of simplex iterations is used as the main metric for comparing algorithms.

Algorithms from the same experiments are compared against each other by means of *performance profiles* on time, number of simplex iterations, and node count. Performance profiles provide an efficient and convenient benchmarking tool for comparing algorithms using measurements provided by various performance metrics [Dolan and Moré, 2002]. They generate a cumulative probability distribution function for the specified performance metric. Performance profiles plot the ratio of one method's performance to the best performance achieved against the percentage of models at which the method attains that ratio or better. This approach eliminates problems such as skewed averages due to a minority of problems dominating the results.

Performance profiles also provide a way of comparing the robustness of algorithms. Robustness is defined here as the ability to solve more models within the specified time limit. While other benchmarking tools may not include failures as part of their performance analysis, performance profiles take this into account. Performance profiles handle robustness by explicitly showing the fraction of models that were not solved within the assigned time limit by a particular algorithm. This can be achieved by looking at the maximum fraction of models (or probability) that were solved by that algorithm, and simply deducting that fraction from 1. Apart from failures, if we take simplex iterations as our performance metric then the performance ratio of one algorithm to the best algorithm for a particular model can be computed using the following equation:

$$r_{p,a} = \frac{t_{p,a}}{\min (t_{p,a} : a \in A)},$$

where r is the performance ratio of a particular algorithm on a problem p , $t_{p,a}$ is the simplex iterations required for that algorithm to solve that problem, and A is the set of all algorithms being compared. The overall performance of an algorithm over all problems in the problem set can be assessed using the following equation:

$$\rho_a(\tau) = \frac{1}{n_p} \text{size}\{ p \in P : r_{p,a} \leq \tau \},$$

where $\rho_a(\tau)$ is the probability for an algorithm $a \in A$ that a performance ratio $r_{p,a}$ is within a factor τ of the best ratio, and n_p represents the total number of problems in a problem set. The function ρ_a is the cumulative probability distribution function for the performance ratio of a specified metric. It can also

represent the fraction of models solved by a particular algorithm at different ratios to best (See Section 7 for examples of performance profiles).

Due to the nondeterministic factors mentioned earlier that may affect the computational time of algorithms running on a multi-core CPU, the time data collected for the experiments here are rounded to the nearest second. Any times that are less than a second are assigned a ratio to best of 1. Rounding to the nearest second also avoids the very large ratio obtained when some of the solution times are very small. It may be true that a method requiring 10 seconds is 20 times slower than a method requiring 0.5 seconds but the difference is not meaningful in practice.

The number of backtracks is another performance metric that is used in some experiments. This shows the number of dives in the B&B tree required before reaching the first feasible solution. Ideally, one would like to see zero backtracks, meaning that a feasible solution was found in the first leaf of the B&B tree. The number of backtracks can also be thought of as the number of infeasible nodes that are solved before a feasible node is reached.

In order to measure the quality of the MIP solution obtained by an algorithm, the *optimality gap*, which is the difference between a known solution and a value that bounds the best possible solution [INFORMS 2011], is captured. The average optimality gaps of the top algorithms presented in this thesis are compared against that of *Baseline_Cplex* in order to evaluate the quality of the first MIP-feasible solution attained by the algorithms.

7. Designing the Best Algorithm

Experiments in this section test the algorithms described in Section 5 in order to identify the best algorithms that apply general disjunction heuristics. These experiments use different subsets of the training set listed in Appendices A.1, A.2, and A.3. Experiments are set up in such a way that all subroutines (i.e. methods that decide on the different aspects that make up a general disjunction algorithm) are the same in all experiments except for the subroutine being studied in a particular experiment.

A typical general disjunction algorithm begins by deciding on the type of disjunction (axis-parallel or general disjunction) to use at the root node. It then decides on conditions under which it will introduce a general disjunction, which requires establishing if there are enough candidate variables to create a general disjunction and determining when the solution progress is stalling while using axis-parallel branching. If introducing a general disjunction, the next step is to decide on which active constraint to choose as the basis for general branching. While there is a defined method for creating a general disjunction based on an inequality constraint, if the chosen branching constraint is an equality, then one must also decide on how to create the 45-degree general disjunction that is approximately perpendicular to that constraint. Lastly, one must decide on the branching direction for the created general disjunction. All of these decisions (or heuristics) are examined in the experiments presented in Sections 7.1 through 7.4. Table 3 summarizes the default algorithms used for all experiments in these sections.

Table 3: Default general branching algorithm used in all training experiments.

	Type of Decision	Default Algorithm
1	Which type of disjunction to use at the root node?	A general disjunction is generated at the root node of B&B tree.
2	When to perform a general disjunction?	Use Algorithm 4 (see Section 5.1) to trigger the inclusion of a general disjunction. There is no limit on the number of candidate variables required to generate a general disjunction.
3	Which active constraint to choose for general branching?	Use Algorithm 8 (see Section 5.3) to decide on the branching constraint.
4	How to construct a 45-degree general disjunction that is approximately perpendicular to an equality constraint?	Switch sign of non-zero coefficients on odd counter. If an odd number of non-zeros exist, then set the first non-zero coefficient in the branching constraint to zero.
5	Which branching direction to use for parallel general disjunctions?	Branch to decrease the value of the disjunction function.
6	Which branching direction to use for perpendicular general disjunctions?	Branch to increase the value of the disjunction function.

Depending on the feature being studied in an experiment, the algorithm used for that feature may change. For example, if the experiment is examining how to choose an active constraint for general branching, then the different algorithms run under this experiment can only change the algorithm used in the third row of Table 3. Choices for algorithms to use include any of Algorithm 5 through Algorithm 10. The same technique is applied to all other experiments that study other aspects of the general disjunction algorithm.

7.1. Experiment 1: Deciding When to Perform a General Disjunction

Heuristics for deciding when to perform a general disjunction are compared in this section. The compared algorithms are run on the subset of training set listed in Appendix A.3. The sub-algorithms used at different decision points in the

general disjunction algorithm remain at default as mentioned earlier, except for the sub-algorithm that decides when to perform a general disjunction. By fixing all parameters of the algorithm and only varying the method used for triggering the inclusion of a general disjunction, one can focus on the effectiveness of the different heuristics.

7.1.1 Recognizing That Axis-Parallel Disjunctions are Performing Poorly

One way of deciding when to use a general disjunction is by determining if the solution progress has stalled while using axis-parallel branching. Different algorithms for monitoring the progress of the MIP solution and triggering the inclusion of a general disjunction were discussed in Section 5.1. The performances of these algorithms are compared here using performance profiles on time, simplex iterations, and node count. Table 4 summarizes the different methods that are compared against each other in the performance profiles given in Figure 7, Figure 8, and Figure 9.

Table 4: Summary of methods for triggering the inclusion of a general disjunction.

	Method Name	Condition Triggering the Inclusion of a General Disjunction	Algorithm Used
1	$(CVC+ISC)>0$	$(cand-var-count + infeas-sum-count) > 0$	Algorithm 2
2	$(CVC+ISC)>5$	$(cand-var-count + infeas-sum-count) > 5$	Algorithm 2
3	$(CVC+ISC)>10$	$(cand-var-count + infeas-sum-count) > 10$	Algorithm 2
4	$(CVC-And-ISC)>3$	$(cand-var-count > 3)$ and $(infeas-sum-count > 3)$	Algorithm 3
5	$(CVC-Or-ISC)>3$	$(cand-var-count > 3)$ or $(infeas-sum-count > 3)$	Algorithm 3
6	$(CVC-Or-ISC)>5$	$(cand-var-count > 3)$ or $(infeas-sum-count > 3)$	Algorithm 3
7	$(CVC-Or-ISC)>10$	$(cand-var-count > 3)$ or $(infeas-sum-count > 3)$	Algorithm 3
8	$(CVRP-And-ISR)>0$	$(cand-var-ratio-progress > 0)$ and $(infeas-sum-ratio-progress > 0)$	Algorithm 4
9	$(CVC-And-ISC)>0$	$(cand-var-count > 0)$ and $(infeas-sum-count > 0)$	Algorithm 3

Figure 7, Figure 8, and Figure 9 compare the different methods for monitoring the progress of the MIP solution while using axis-parallel disjunctions. At some point in the MIP solution, any of the methods provided in Table 4 may establish that axis-parallel branching is performing poorly and trigger the inclusion of a single general disjunction to boost the performance. Since all of these methods are run on a subset of models from the training set (models solved in less than an hour with *Baseline_Cplex*), robustness is an important property that must be taken into account. Looking at Figure 8 (since simplex iterations is the main performance metric for comparison), one can see that the most robust methods are $(CVC+ISC)>5$, $(CVC+ISC)>10$, $(CVC-And-ISC)>3$, and $(CVC-Or-ISC)>10$. At a ratio to best of 4, at which the highest fraction of models is reached, these methods solve approximately 95% of the models within a time

limit of 2 hours (as mentioned in Section 6), whilst the other methods solve less than 90% of the models at that ratio. The reason is that the top methods restrict the use of general disjunctions more than the other methods. As stated in Section 5.1, the over-use of general disjunctions can counteract their gains, hence algorithms such as $(CVC\text{-}And\text{-}ISC)>0$, which trigger the use of general disjunctions anytime the number of candidate variables and infeasibility sum increase, are very inefficient. The average number of general disjunctions generated using the top four methods over all models listed in Appendix A.3 is less than 100 per MIP model, in contrast to an average of more than 300 for $(CVC\text{-}And\text{-}ISC)>0$.

The $(CVRP\text{-}And\text{-}ISRP)>0$ method, which uses ratios instead of counts for monitoring the progress of the MIP solution, did not perform very well compared to the other methods. This could result from the fact that at smaller numbers of candidate variables, the change in ratio may prematurely trigger the use of general disjunctions. For example, if the number of candidate variables at the parent node is 2, and 4 at the child node, then *cand-var-ratio* has a value of 1 (see Section 5.1 for an example), which could potentially trigger the use of a general disjunction even though the change in the number of candidate variables is not remarkable.

$(CVC\text{-}And\text{-}ISC)>3$ clearly performs better than $(CVC\text{-}Or\text{-}ISC)>3$, as it solves 10% more models than the latter method at a ratio to best of 4. Even though both methods use the same algorithm (Algorithm 3), nonetheless, $(CVC\text{-}And\text{-}ISC)>3$ restricts the use of general disjunctions further than $(CVC\text{-}Or\text{-}ISC)>3$

because it triggers the inclusion of a general disjunction only if both the number of candidate variables and the infeasibility sum stay the same or increase from their previous values at the parent node three times in a row. On the other hand, $(CVC-Or-ISC)>3$ triggers the use of general disjunctions if either of these measures change in the same manner more than three times in a row. The number of candidate variables and the infeasibility sum are interrelated performance metrics that work effectively together, which is why adding *cand-var-count* and *infeas-sum-count* is another successful way of monitoring the performance of the MIP solution.

Figure 7: Experiment 1 performance profile on computational time for detecting when axis-parallel disjunctions are performing poorly.

Figure 8: Experiment 1 performance profile on simplex iterations for detecting when axis-parallel disjunctions are performing poorly.

Figure 9: Experiment 1 performance profile on node count for detecting when axis-parallel disjunctions are performing poorly.

7.1.2. Limiting the Number of General Disjunctions

One of the main advantages of general disjunctions is that they impact many candidate variables at once. Consequently, it is only worth including one if there are many candidate variables to be impacted. If the number of candidate variables at a particular node is fairly small, then the cost of inserting and maintaining a general disjunction might outweigh the benefits of using it. Section 5.1 mentioned adding a lower limit on the number of candidate variables before including general disjunctions. Experiments in this section compare the performance of some of the best methods obtained from Section 7.1.1 with and without adding requirements on the minimum number of candidate variables necessary for triggering the use of general disjunctions.

To examine the effectiveness of adding a global requirement on the minimum number of candidate variables, the models on which the methods perform well (i.e. models that are solved significantly faster by one method in comparison to other methods) from the top four methods obtained from Section 7.1.1 are examined closely. Similarly the models on which the methods perform poorly (i.e. models that are solved significantly slower by one method in comparison to other methods) from the same set of methods are analysed in detail. One way of analysing the performance of these methods on such models is to study the pattern of general disjunctions triggered at various points in the B&B tree. The numbers of candidate variables at the upper nodes of the B&B tree are typically higher than those at the lower nodes of the tree. Hence, to examine the pattern of general disjunctions at different levels of the B&B tree, a

histogram that specifies the range of numbers of candidate variables is defined, as shown in Figure 10. The number of general disjunctions used in a MIP solution is then considered for each of these histogram bins. Typically one would like to see the number of general disjunctions decrease as the number of candidate variables decreases down the B&B tree. This is because the cost of creating general disjunctions at the lower levels of the tree when there are not many candidate variables is not worth the benefits gained at such levels. Table 5 summarizes the simplex iterations measure obtained from running the top four methods on the models on which they perform well. The "TL" values in the table indicate that a time limit of two hours was exceeded, and the highlighted cells represent those models that were solved relatively quickly by a specific method.

Table 5: Simplex iterations of models that are executed relatively quickly by the best methods that trigger the inclusion of a general disjunction.

	Simplex Iterations			
	(CVC+ISC)>5	(CVC+ISC)>10	(CVC-And-ISC)>3	(CVC-Or-ISC)>10
30n20b8.mps	1708883	1708883	TL	2207435
acc-tight5.mps	917660	917660	399503	917660
gmvt-75-50.mps	48532	48532	12385	48532
gmvt-77-40.mps	29323	29323	172986	29323
neos-1171737.mps	127180	127180	6320	127180
b2c1s1.mps	93597	93597	86630	27527
mzzv11.mps	98923	98923	96886	98923
csched007.mps	399059	399059	14975345	12496436
lotsize.mps	TL	5916936	99198	110940

Figure 10 through Figure 13 present the number of general disjunctions triggered by each of the best methods for the models that were solved the fastest by that method. Since (CVC+ISC)>5 and (CVC+ISC)>10 methods generate on average 1.1 and 1.4 general disjunctions per model, respectively, this explains

why the numbers of general disjunctions in Figure 10 and Figure 11 do not exceed 1.

Figure 10: A histogram of the number of general disjunctions triggered by $(CVC+ISC)>5$ at different levels in the B&B tree for 30n20b8.mps, gmut-77-40.mps, and csched007.mps.

Figure 11: A histogram of number of general disjunctions triggered by $(CVC+ISC)>10$ at different levels in the B&B tree for 30n20b8.mps, gmut-77-40.mps, and csched007.mps.

As can be seen in Figure 10 and Figure 11, all the general disjunctions used in the MIP solution were added nearer the root node as they were

generated in nodes where the number of candidate variables was greater than 60. The main thing to note is the fact that there were no general disjunctions generated at the lower levels of the B&B tree, which may be why these models were solved the fastest by $(CVC+ISC)>5$ and $(CVC+ISC)>10$.

Figure 12: A histogram of the number of general disjunctions triggered by $(CVC-And-ISC)>3$ at different levels in the B&B tree for acc-tight5.mps, gmut-75-50.mps, neos-1171737.mps, and mzzv11.mps.

Figure 13: A histogram of the number of general disjunctions triggered by $(CVC-Or-ISC)>10$ at different levels in the B&B tree for gmut-77-40.mps, b2c1s1.mps, and lotsize.mps.

Figure 12 and Figure 13 illustrate as well the idea that the number of general disjunctions decreases as we descend the B&B tree for models which use many general disjunctions. Once again, this may explain the performance gains attained by solving these models using these limits. Furthermore, the pattern of general disjunctions triggered at various points in the B&B tree for those models that were solved the slowest by the different methods is also observed. Since $(CVC+ISC)>5$ and $(CVC+ISC)>10$ methods do not trigger many general disjunctions per model, histograms that show the pattern of general disjunctions for the models on which these methods perform poorly are not very useful.

Figure 14 and Figure 15 provide histograms that show the pattern of general disjunctions triggered by $(CVC-And-ISC)>3$, and $(CVC-Or-ISC)>10$, respectively, for the three models on which they perform poorly.

Figure 14: A histogram of the number of general disjunctions triggered by $(CVC-And-ISC)>3$ for the models on which it performs poorly.

Figure 15: A histogram of the number of general disjunctions triggered by *(CVC-Or-ISC)>10* for the models on which it performs poorly.

Figure 14 and Figure 15 show that the distribution of general disjunctions triggered at different points in the MIP solution is irregular. Many of the general disjunctions were generated by these methods at the lower levels of the B&B tree. In fact, a significantly higher number of general disjunctions are triggered by these methods while solving the models on which they perform poorly, in comparison to the number general disjunctions triggered by the same methods while solving the models on which they perform well in Figure 12 and Figure 13.

In order to determine the best global limit to use on the minimum number of candidate variables required to trigger the use of a general disjunction, three different values were tried: i) 20 candidate variables ii) 40 candidate variables and iii) 60 candidate variables. A global limit of more than 60 candidate variables was not used as most of the general disjunctions were triggered for those models on which the algorithms performed well when there were 60 candidate variables or higher as shown in Figure 10 through Figure 13. Performance profiles on

simplex iterations for each of the top methods obtained from Section 7.1.1, comparing the performances of these methods with different requirements on the minimum number of candidate variables are provided in Figure 16 through Figure 19.

Since the average number of general disjunctions triggered by $(CVC+ISC)>5$ and $(CVC+ISC)>10$ methods is small, one should not expect significant performance gains when a prerequisite on the minimum number of candidate variables is enforced. This can be seen in the performance profiles provided in Figure 16 and Figure 17, which compare the performance of these methods with different requirements on the minimum number of candidate variables. Given that the performances of these methods are similar, even when various global requirements are imposed, the two methods that solve the largest fraction of models in each of the performance profiles are chosen for comparison with the two best methods from Figure 18 and Figure 19. Looking closely at Figure 16 and Figure 17, the two best methods that solve the highest fractions of models (before reaching a ratio to best of 10) are $(CVC+ISC)>5$ (Cand Vars > 60) and $(CVC+ISC)>10$ (Cand Vars > 60), correspondingly.

Figure 16: Experiment 1 performance profile on simplex iterations for determining the best global requirement on the minimum number of candidate variables for $(CVC+ISC)>5$.

Figure 17: Experiment 1 performance profile on simplex iterations for determining the best global requirement on the minimum number of candidate variables for $(CVC+ISC)>10$.

Noticeable improvements can be seen in the performance profiles for $(CVC\text{-}And\text{-}ISC)>3$ (Figure 18) and $(CVC\text{-}Or\text{-}ISC)>10$ (Figure 19) with different global requirements on the minimum number of candidate variables. Comparing $(CVC\text{-}And\text{-}ISC)>3(\text{No Limit on Cand Vars})$ with $(CVC\text{-}And\text{-}ISC)>3(\text{Cand Vars} > 60)$ at a ratio to best of 2 in Figure 18, $(CVC\text{-}And\text{-}ISC)>3(\text{Cand Vars} > 60)$ is able to solve 96% of the models within the time limit, in contrast to $(CVC\text{-}And\text{-}ISC)>3(\text{No Limit on Cand Vars})$ which is only able to solve 91% of the models at that ratio. Furthermore, the overall robustness of $(CVC\text{-}And\text{-}ISC)>3(\text{Cand Vars} > 60)$ is higher than without adding a global requirement on the minimum number of candidate variables. Looking at these performance profiles, one can conclude that the best global requirement to enforce with $(CVC\text{-}And\text{-}ISC)>3$ is a minimum requirement of 60 candidate variables.

Figure 18: Experiment 1 performance profile on simplex iterations for determining the best global requirement on the minimum number of candidate variables for $(CVC\text{-}And\text{-}ISC)>3$.

The best global requirement on the minimum number of candidate variables to use with $(CVC-Or-ISC)>10$ is either 40 or 60 candidate variables, as the performance profile given in Figure 19 shows that the best two methods (i.e. most robust) are $(CVC-Or-ISC)>10(Cand\ Vars > 40)$ and $(CVC-Or-ISC)>10(Cand\ Vars > 60)$. The performances of $(CVC-Or-ISC)>10(No\ Limit\ on\ Cand\ Vars)$ and $(CVC-Or-ISC)>10(Cand\ Vars > 20)$ are almost exactly the same as the performance profile graphs for both methods settle on top of each other. The highest fraction of models solved by $(CVC-Or-ISC)>10(Cand\ Vars > 60)$ is 0.99, whereas the highest fraction of models solved by $(CVC-Or-ISC)>10(No\ Limit\ on\ Cand\ Vars)$ is 0.95. Since $(CVC-Or-ISC)>10(Cand\ Vars > 60)$ is slightly more robust than $(CVC-Or-ISC)>10(Cand\ Vars > 40)$, it is chosen as one of the top methods to compare with other top methods in Figure 20.

Figure 19: Experiment 1 performance profile on simplex iterations for determining the best global requirement on the minimum number of candidate variables for $(CVC-Or-ISC) > 10$.

The performance profile in Figure 20 compares the top methods obtained from the performance profiles in Figure 16 through Figure 19. All the top methods compared in Figure 20 specify a minimum requirement of 60 candidate variables to trigger the use of general disjunctions. With such a condition in place, the top two methods (most robust) in Figure 20 are $(CVC-And-ISC) > 3 (Cand Vars > 60)$ and $(CVC-Or-ISC) > 10 (Cand Vars > 60)$. At a ratio to best of 2, $(CVC-And-ISC) > 3 (Cand Vars > 60)$ and $(CVC-Or-ISC) > 10 (Cand Vars > 60)$ are both able to solve a higher fraction of models than $(CVC+ISC) > 5 (Cand Vars > 60)$ and $(CVC+ISC) > 10 (Cand Vars > 60)$, and thus are selected as two of the best methods for inclusion in the later experiments.

Figure 20: Experiment 1 performance profile on simplex iterations for determining the best method for triggering the inclusion of a general disjunction with a global requirement on the minimum number of candidate variables.

7.1.3 Adding a General Disjunction After the Root Node

Any decisions that are taken higher up in the B&B tree will generally affect the performance (either in a good or bad way) of subsequent nodes in the tree. The default algorithm used for the first type of decision in Table 3 includes a general disjunction at the root node (right after attaining the LP-relaxation solution of the root node). This is a special case as there is no evidence that axis-parallel branching is performing poorly (since there has been no opportunity to build it up), and yet the largest number of candidate variables is at the root node.

An experiment that compares the performance of method (CVC-And-ISC)>3 from Section 7.1.1 with and without adding a general disjunction at the root node

is presented here. Figure 21, Figure 22, and Figure 23 show performance profiles for different metrics in order to determine the effectiveness of adding a general disjunction at the root node.

In Figure 22, the performances of $(CVC\text{-}And\text{-}ISC)>3$ with and without adding a general disjunction at the root node are very similar; nevertheless, adding a general disjunction at the root node makes the method slightly more robust. Note that there is no global requirement on the minimum number of candidate variables. If a global requirement is to be specified, then a general disjunction would not be used at the root node unless this requirement is satisfied. The effects of such an implementation is studied in Section 7.5.

Figure 21: Experiment 1 performance profile on computational time for comparing the performance of $(CVC\text{-}And\text{-}ISC)>3$ with and without adding a general disjunction at the root node.

Figure 22: Experiment 1 performance profile on simplex iterations for comparing the performance of (CVC-And-ISC)>3 with and without adding a general disjunction at the root node.

Figure 23: Experiment 1 performance profile on node count for comparing the performance of (CVC-And-ISC)>3 with and without adding a general disjunction at the root node.

7.2. Experiment 2: Choosing an Active Constraint for General Branching

When a general disjunction is to be added, the next thing is to decide on the branching constraint that will be approximated by the general disjunction. Section 5.3 discussed the various branching-constraint selection schemes that can be used for this purpose. Different algorithms that test these heuristics for choosing the branching constraint are compared in this section. The compared algorithms are run on the subset of the training set listed in Appendix A.3.

Some of the schemes described in Section 5.3 perform some sort of normalization in order to adjust the relative weight of each active constraint. However, experiments show that the unnormalized algorithms perform comparatively better than the normalized ones. This is likely because Cplex, the MIP solver, uses advanced scaling techniques to adjust the constraint coefficients. For this reason, decisions based on normalization using the original coefficients may be ineffective.

Because the normalized branching-constraint selection schemes do not provide any performance gains over the unnormalized schemes, normalization schemes are not used here. The experiment conducted here mainly focuses on identifying the best branching-constraint selection schemes, by testing the different algorithms that can be used for the third type of decision in Table 3. All other types of decisions still use the assigned default algorithms. Performance profiles on time, simplex iterations, and node count compare the various branching-constraint selection methods provided in Table 6. Algorithm 9 and

Algorithm 10 from Section 5.3 are not compared here as they are the normalized versions of Algorithm 7.

Table 6: Summary of branching constraint selection methods compared in Experiment 2.

	Name on Performance Profile	Full Method Name	Description	Algorithm Used
1	CVS-FGS	<i>candVarSum-feasGapSum</i>	See Table 2	Algorithm 5
2	IVS-SIC	<i>intVarSum-sumIntCoeffs</i>	Choose the active constraint with the largest number of integer variables. Break ties between equally-ranked constraints using the highest sum of integer-variable absolute coefficients (unnormalized)	Algorithm 6 (unnormalized version)
3	SCC-FGS	<i>sumCandCoeffs-feasGapSum</i>	See Table 2	Algorithm 7
4	CVS-SCC	<i>candVarSum-sumCandCoeffs</i>	Choose the active constraint with largest number of candidate variables. Break ties between equally-ranked constraints using the highest sum of candidate-variable absolute coefficients (unnormalized)	Algorithm 8 (unnormalized version)

The performance profiles given in Figure 24, Figure 25, and Figure 26 compare the branching-constraint selection methods summarized in Table 6. Using simplex iterations as our main metric for comparing algorithms, the performance profile given in Figure 25 shows that the most robust (within a ratio to best of 10) branching-constraint selection method is *IVS-SIC*. *IVS-SIC* compares both the number and coefficient magnitudes of integer variables in the

active constraints. These two measures are static since they do not change as the MIP solution progresses. On the other hand, *CVS-SCC*, which compares the number and coefficient magnitudes of candidate variables in the active constraints, uses dynamic measures that change as the set of candidate variables changes in the MIP solution. One can note from Figure 25 that the performances of *CVS-FGS* and *SCC-FGS* are very similar as they both solve approximately 92% of the models, in comparison to *IVS-S/C* (best method) which solves around 99% of the models. Both *CVS-FGS* and *SCC-FGS* use the *feasGapSum* measure to break the ties between equally-ranked active constraints. This measure may be good at breaking ties, as will be discussed in Section 7.2.1, but it is not the best measure for deciding on the active constraint that has the most impact on achieving integer feasibility quickly.

Branching-constraint selection methods that use dynamic measures based on the candidate variables in a MIP solution work very well at the higher levels of the B&B tree where there are many candidate variables present. As we descend the B&B tree, the number of candidate variables decreases and hence becomes an ineffective measure for assessing the impact of active constraints. This explains why static measures, such as the number of integer variables, provide better robustness than dynamic measures which are rather ineffective at lower levels of the B&B tree. Enforcing requirements on the minimum number of candidate variables required to trigger a general disjunction, as mentioned in Experiment 1 in Section 7.1, would likely make dynamic measures like the number of candidate variables much more effective. This is because such

measures would only be used when there are sufficient numbers of candidate variables in the LP-relaxation solution. Experiments that compare the effectiveness of *IVS-S/C* and *CVS-SCC* after adding global requirements on the minimum number of candidate variables are provided in Section 7.5.

In addition to using the number of integer or candidate variables to choose the active constraint that has the highest impact on MIP-feasibility, the sum of integer-variable or candidate-variable absolute coefficients is another useful measure. Assuming that the MIP model is reasonably scaled (either by the user or the solver), then larger absolute coefficients indicate a larger impact of the associated variable on the constraint. As a result, branching approximately parallel or perpendicular to the active constraint with the highest sum of integer-variable or candidate-variable absolute coefficients will most likely force the values of those variables to change, consequently speeding the process of attaining a MIP-feasible solution.

Figure 24: Experiment 2 performance profile on time for choosing an active constraint for general branching.

Figure 25: Experiment 2 performance profile on simplex iterations for choosing an active constraint for general branching.

Figure 26: Experiment 2 performance profile on node count for choosing an active constraint for general branching.

7.2.1. A Closer Look at the Number of Tied Branching Constraints

All the branching-constraint selection methods compared in Section 7.2 use two criteria to rank the active constraints. The first criterion initially ranks the active constraints, selects the top-ranked one(s), thereby filtering out those constraints with the least impact on the MIP solution. The second criterion helps break the ties between multiple top-ranked active constraints. This section looks closely at the number of ties among the top-ranked active constraints that remain even after applying a second ranking criterion.

The average percentage of tied top-ranked active constraints is used to compare the number of ties resulting from the different branching-constraint selection methods. This number can be calculated as follows:

Average % of Tied Top-Ranked Active Constraints

$$= \frac{\sum_{\text{all model}} \sum_{\text{all gen. disj.}} \frac{\text{Number of tied top-ranked active constraints}}{\text{Total number of active constraints}}}{N}$$

where N represents the total number of models run in the experiment (i.e. $N = 100$ models from Appendix A.3). Table 7 summarizes the results for each of the branching-constraint selection methods.

Table 7: Average percentage of tied top-ranked active constraints for each branching-constraint selection method.

Branching-Constraint Selection Method	Avg. % of Tied Active Constraints	% of Models with No Active-Constraint Ties	% of Models with all Active Constraints Tied
<i>CVS-FGS</i>	3	68	1
<i>IVS-SIC</i>	10	47	2
<i>SCC-FGS</i>	2	76	1
<i>CVS-SCC</i>	7	44	2

One can note from Table 7 that *CVS-FGS* and *SCC-FGS*, which both use the *feasGapSum* measure to break the ties between equally-ranked active constraints, have the lowest percentages of tied constraints. The *feasGapSum* measure uses the LP-relaxation values of candidate variables at a node in order to break the ties between top-ranked active constraints. Although *CVS-FGS* and *SCC-FGS* methods on average have the lowest percentages of tied constraints, nonetheless, these methods do not perform as well as *IVS-SIC* and *CVS-SCC* when deciding on the active constraint for general branching.

Breaking ties between equally-ranked active constraints by using the sum of variable (either integer or candidate) absolute coefficients, as done by *IVS-SIC* and *CVS-SCC* methods, may be ineffective when all active constraints have very similar coefficient magnitudes. This problem especially arises if most or all of the active constraints' absolute coefficients are binary (1 or 0). However, the

percentage of models in which all active constraints are ranked equally is on average 1-2% for all branching-constraint selection methods. This percentage is considerably lower than the percentage of models which do not result in any active-constraint ranking ties (44-76%).

7.3. Experiment 3: Creating a 45-Degree Perpendicular General Disjunction

The branching constraint chosen by any of the selection schemes studied in Section 7.2 could be either an inequality or equality constraint. If the chosen constraint is an inequality constraint, then an approximately parallel general disjunction is created using that constraint (see Algorithm 11 in Section 5.4). On the other hand, if the chosen constraint is an equality constraint, then a general disjunction that is as perpendicular as possible to that constraint is created.

Section 5.4.1 described some of the different ways of creating an approximately perpendicular 45-degree general disjunction. There are two essential decisions that must be made: (i) if there exists an odd number of nonzero coefficients in the branching constraint, then one must decide on which coefficient to set to zero in the disjunction equation, and (ii) how to set the signs on the remaining nonzero coefficients in the disjunction equation. An experiment that compares the different schemes, discussed in Section 5.3.1, for making these two decisions is conducted in this section.

The algorithm used for the fourth type of decision in Table 3 is varied in this experiment in order to determine the best method for creating an approximately perpendicular general disjunction. The default algorithms used for all other types

of decisions in Table 3 are left as is. The methods compared here are run on the subset of the training set listed in Appendix A.1, which require the MIP models to contain at least one equality constraint. After the different methods are run, the models listed in Appendix A.1 are filtered even further, such that only those models in which a perpendicular general disjunction is triggered are used for generating the performance profiles presented here (See Appendix A.6 for list of filtered models).

If one variable coefficient must be set to zero, it makes sense to choose the one that has the least impact on the branching equality constraint. For the first part of Experiment 3, different least-impact-variable-selection methods are compared against each other. Signs of nonzero coefficients in the branching constraint are switched on odd counter in the disjunction equation, as shown in Algorithm 12. Table 8 summarizes the different least-impact-variable-selection methods that could be used for Step 1 of Algorithm 12.

Table 8: Summary of least-impact-variable-selection methods compared in Experiment 3.

	Method Name on Performance Profile	Description
1	<i>SII-intVar</i>	For all variables with non-zero coefficients in the branching-constraint, choose the first integer variable with the smallest integer infeasibility. An integer infeasibility of zero is possible for integer variables that have taken on an integer value.
2	<i>SC-var</i>	For all variables with non-zero coefficients in the branching-constraint, choose the variable with the smallest coefficient.
3	<i>SC(realVar-NCintVar-candVar)</i>	For all variables with non-zero coefficients in the branching-constraint, choose the real variable with the smallest coefficient. If no real variables exist in the constraint, then choose the non-candidate integer variable with the smallest coefficient. If constraint only consists of candidate variables then choose the one with the smallest coefficient.
4	<i>SC(realVar-NCintVar)-LA(candVar)</i>	For all variables with non-zero coefficients in the branching-constraint, choose the real variable with the smallest coefficient. If no real variables exist in the constraint, then choose the non-candidate integer variable with the smallest coefficient. If constraint only consists of candidate variables then choose the one that is present in the least number of active constraints.

The least-impact-variable-selection methods listed in Table 8 are based on choosing the variable having the least impact on the LP-relaxation solution, and hence setting the coefficient of that variable to zero in the disjunction equation should have the least effect. These methods are compared in the performance profiles (on time, simplex iterations, and node count) presented in Figure 27, Figure 28, and Figure 29.

There are 43 models (listed in Appendix A.6) out of the 100 equality-containing MIP models in which the use of a perpendicular general disjunction is triggered. The performances of each of the least-impact-variable-selection

methods with respect to these models are plotted on the performance profiles in Figure 27, Figure 28, and Figure 29. As can be seen in Figure 28, *SC(realVar-NCintVar)-LA(candVar)* method is the most robust as it solves the highest fraction of models (0.88, which is the highest fraction of models solved by any of the methods) before reaching a ratio to best of 5. At a ratio to best of 5, *SII-intVar* and *SC(realVar-NCintVar-candVar)* solve approximately 80% of the models within the time limit, which is lower than the fraction of models solved by *SC-var* at that ratio (0.86).

Since a branching constraint is chosen such that it has the most impact on the candidate variables, hence the variables that make up the branching-constraint equation are mostly, if not all, integer variables. This is why least-impact-variable-selection methods that are only based on choosing a real variable in the branching-constraint equation need to account for constraints that do not have real variables. Methods *SC(realVar-NCintVar-candVar)* and *SC(realVar-NCintVar)-LA(candVar)* take that into account. Since the first two conditions in these methods are the same, the differences in performances depend on the last condition which is triggered when the branching constraint has only candidate variables.

Choosing the candidate variable that is present in the least number of active constraints, as chosen by *SC(realVar-NCintVar)-LA(candVar)*, clearly has less impact on the MIP solution than choosing the candidate variable that has the smallest coefficient in the branching constraint, as chosen by *SC(realVar-NCintVar-candVar)*. This is because a candidate variable that is present in many

active constraints has a great influence on the MIP solution as the value of such a candidate variable is forced to change in order to satisfy the various constraints. Consequently, setting the coefficient of the candidate variable that is present in the least number of active constraints to zero in the perpendicular disjunction function has the least impact on the MIP solution, and hence is a more effective method than $SC(realVar-NCintVar-candVar)$.

A "45-degee" perpendicular disjunction is created such that it is as perpendicular as possible to an active equality constraint and will thus force multiple candidate variables to change at once. Since the performance of $SC-var$ is generally better than $SC(realVar-NCintVar-candVar)$, this means that choosing the smallest coefficient variable in the branching constraint regardless of the type of variable has less impact on the MIP solution. Setting the coefficient of the smallest coefficient variable in the branching constraint to zero in the disjunction equation should help provide a better approximately 45-degree perpendicular general disjunction.

Figure 27: Experiment 3 performance profile on time for choosing the variable with the least impact on the MIP solution.

Figure 28: Experiment 3 performance profile on simplex iterations for choosing the variable with the least impact on the MIP solution.

Figure 29: Experiment 3 performance profile on node count for choosing the variable with the least impact on the MIP solution.

Given that the performances of *SC(realVar-NCintVar)-LA(candVar)* and *SC-var* are very close in Figure 28, a performance profile on simplex iterations for those models in which only perpendicular general disjunctions were used is provided in Figure 30. This performance profile compares each of the least-impact-variable-selection methods for a sub-set of 28 equality-containing MIP models in which no parallel general disjunctions were generated (See Appendix A.6 for set of models). One can now notice that the performance of *SC-var* exceeds that of *SC(realVar-NCintVar)-LA(candVar)*. At a ratio to best of 2, *SC-var* solves 89% of the models in comparison to *SC(realVar-NCintVar)-LA(candVar)* which solves only 83% of the models at that ratio. The performance

of *SC-var* is also better than *SC(realVar-NCintVar)-LA(candVar)* from the node count perspective, as shown in the performance profile in Figure 29.

Figure 30: Experiment 3 performance profile on simplex iterations for choosing the variable with the least impact on the MIP solution (for only models that used perpendicular general disjunctions).

We next examine the performance of *SC-var* when switching signs on odd or even counters. A performance profile on simplex iterations is provided in Figure 31. This performance profile is generated for the 43 models that used at least one perpendicular general disjunction. The performances of the compared methods for the subset of 28 models that did not use any parallel general disjunctions provides a very similar performance profile, and thus it is not shown here. As can be seen in Figure 31, both ways of creating a 45-degree perpendicular general disjunction have very similar performances. However, one

can note that by switching signs on even counter, *SC-var* solves around 60% of the models at a faster rate than by switching signs on odd counter, and hence is selected as the top method for creating a perpendicular general disjunction.

Figure 31: Experiment 3 performance profile on simplex iterations for creating a 45-degree perpendicular disjunction using *SC-var* by switching signs on odd or even counts.

7.4. Experiment 4: Selecting the Branching Direction For General Disjunctions

Experiments that help determine the best branching direction for parallel and perpendicular general disjunctions are presented in this section. Section 7.4.1 focuses on experiments that select the branching direction as well as the offset for parallel general disjunctions, while Section 7.4.2 presents experiments that select the branching direction for perpendicular general disjunctions.

7.4.1. Experiment 4.A: Selecting the Branching Direction and Offset For Parallel General Disjunctions

The branching-direction selection schemes for parallel disjunctions that were discussed in Section 5.5 are compared here (i.e. the fifth type of decision in Table 3). The various methods are run on the subset of MIP models listed in Appendix A.2. Furthermore, in order to focus on the performances of the branching-direction selection schemes, the 72 models that only utilized parallel general disjunctions as part of their solutions, which are listed in italics in Appendix A.2, are used for generating the performance profiles in this section. Table 9 lists the different branching-direction selection methods that are compared in the performance profiles provided in Figure 32, Figure 33, and Figure 34.

Table 9: Summary of branching-direction selection methods compared in Experiment 4.A.

	Method Name on Performance Profile	Branching Direction Description
1	<i>C-LP</i>	Closer to the LP-relaxation optimum point.
2	<i>F-LP</i>	Farther from LP-relaxation optimum point.
3	<i>Dec-DF</i>	Decrease the value of the disjunction function.
4	<i>Inc-DF</i>	Increase the value of the disjunction function.
5	<i>Vio-BC</i>	Violating direction of the branching constraint.
6	<i>Sat-BC</i>	Satisfying direction of the branching constraint.

Performance profiles on time, simplex iterations, and node count for the subset of 72 MIP models are shown in Figure 32, Figure 33, and Figure 34, respectively. Looking at Figure 33, one can see that the most robust branching-direction selection methods are *F-LP*, *Dec-DF*, and *Sat-BC*, as they solve 96% of

the models before reaching a ratio to best of 5. Seeing that the performance of *Dec-DF* is mostly overlapping with the performance of *Sat-BC*, this could mean that most of the branching constraints selected in the set of models were of type " $\leq$ ". If the opposite was true (i.e. most branching constraints were of type " $\geq$ "), then the performance of *Inc-DF* would be very similar to that of *Sat-BC*. Therefore, *Sat-BC* is chosen to represent the best of the two branching directions.

In the spirit of Pryor and Chinneck's [2011] idea of branching to force change, *F-LP* and *Sat-BC* methods are known to force the greatest number of candidate variables to change simultaneously, leading to an integer feasible solution faster (see Section 5.5 for more details). Consequently, they are chosen as the top branching-direction selection methods for parallel general disjunctions.

The average percentage of *cuts* is another measure that is collected. A specific branch of a general disjunction is considered to be a "cut" if this branch eventually turns out to be MIP-infeasible. The highest average percentage of cuts over all branching-direction selection methods is 31%, which is attained by branching in the violating direction of the branching constraint (*Vio-BC*). On the other hand, the smallest average percentage of cuts is 9%, which is attained by branching in the satisfying direction of the branching constraint (*Sat-BC*).

Figure 32: Experiment 4.A performance profile on time for selecting the branching direction for parallel general disjunctions.

Figure 33: Experiment 4.A performance profile on simplex iterations for selecting the branching direction for parallel general disjunctions.

Figure 34: Experiment 4.A performance profile on node count for selecting the branching direction for parallel general disjunctions.

The main advantage of branching in the satisfying direction of the branching constraint is that it constricts the feasible region and thus forces many candidate variables to change at once. As mentioned in Section 5.5.1, branching in the satisfying direction of the branching constraint may be ineffective if the branching constraint is a "45-degree" inequality. Offsetting the RHS value of a parallel general disjunction by one unit may help overcome this problem. Another experiment is conducted to compare the performance of *Sat-BC* with and without offsetting the RHS of the 45-degree parallel general disjunction by one unit.

The performance profile in Figure 35 compares the performance of *Sat-BC* with and without offsetting the RHS of the parallel disjunction by one unit. The performances of both methods are very similar. This could be because: i) not

many branching constraints have coefficients that are the same as the disjunction function, or ii) the feasible region bounded by the branching constraint may be very narrow, and thus branching in the satisfying direction with an offset of one unit may occasionally lead to an infeasible LP sub-problem at the child node. However, since adding an offset of one unit with *Sat-BC* makes the algorithm solve around 58% of the models at a faster rate than without adding the offset, *Sat-BC* (with offset) is selected as our top branching-direction selection method along with *F-LP*.

Figure 35: Experiment 4.A performance profile on simplex iterations for comparing the performance of *Sat-BC* with and without offsetting the RHS of the disjunction by one unit.

7.4.2. Experiment 4.B: Selecting the Branching Direction For Perpendicular General Disjunctions

This section compares the different branching-direction selection schemes for perpendicular general disjunctions. The performance profiles presented in this section compare the various methods on a subset of 27 models from Appendix A.6 in which only perpendicular general disjunctions were used. The experiment conducted here varies the algorithm used for the sixth type of decision in Table 3 in order to determine the best branching direction for perpendicular general disjunctions. The first four branching-direction selection methods in Table 9 are compared for perpendicular general disjunctions in the performance profiles provided in Figure 36, Figure 37, and Figure 38. Branching in the satisfying or violating direction of the branching constraint (i.e. the fifth and sixth branching-direction selection methods in Table 9) cannot be applied to perpendicular general disjunctions as they can only be used if the branching constraint is an inequality.

Looking at Figure 37, one can note that the best two branching-direction selection methods for perpendicular general disjunctions are *F-LP* and *Inc-DF*. Using Pryor and Chinneck's [2011] idea of branching to force change, branching farther from the LP-relaxation optimum point tends to force the greatest number of candidate variables to change and thus speeds the process of attaining a MIP-feasible solution.

Pryor and Chinneck [2011] discussed the impact of multiple choice constraints on the branching-direction decision for axis-parallel disjunctions. A multiple-choice constraint takes the form $\sum_{i=1}^n x_i = 1$ or $\sum_{i=1}^n x_i \leq 1$, where all of

the n variables are binary. Branching in the up direction for one of the variables in the constraint is most desirable as it forces all variables in the constraint to take on integer values, and hence achieves MIP-feasibility quickly. On a similar note, branching to increase or decrease the value of the disjunction function may influence the speed of attaining the first feasible solution depending on the type of branching constraint (i.e. whether the branching constraint is a multiple-choice constraint or not). The branching constraint is a multiple-choice constraint in 56% of the disjunctions in models that only used perpendicular general disjunctions; which explains the difference in performance between *Dec-DF* and *Inc-DF*. Furthermore, the performance of these branching-direction selection methods will depend on the coefficient signs of candidate variables in the disjunction function, as mentioned in Section 5.5.

At a ratio to best of 2 in the performance profile in Figure 37, *F-LP* solves around 90% of the models in comparison to *Inc-DF* which solves only 80% of the models. Moreover, *F-LP* is able to solve all of the models within the time limit, whereas *Inc-DF* solves a maximum of 98% of models (at a ratio to best of 4). As a result, *F-LP* is ranked as the top branching-direction selection method for perpendicular general disjunctions.

Figure 36: Experiment 4.B performance profile on time for selecting the branching direction for perpendicular general disjunctions.

Figure 37: Experiment 4.B performance profile on simplex iterations for selecting the branching direction for perpendicular general disjunctions.

Figure 38: Experiment 4.B performance profile on node count for selecting the branching direction for perpendicular general disjunctions.

7.5. Experiment 5: Constructing the Best Algorithms

Sections 7.1 through 7.4 focused on the different decisions in the general disjunction algorithm that are summarized in Table 3. The best one or two methods for making each decision, as determined by Experiments 1- 4, are combined here to form complete algorithms, from which the best are chosen as the final algorithms that are compared against *Baseline_Cplex* (state-of-the-art MIP solver). The algorithms compared here are run on the subset of MIP models listed in Appendix A.3. Table 10 summarizes the top methods for making each decision.

Table 10: Summary of top methods for making each decision in a complete algorithm.

	Type of Decision	Method Used
1	Which type of disjunction to use at the root node?	A general disjunction is generated at the root node
2	When to perform a general disjunction?	2.A: $(CVC-And-ISC) > 3(Cand Vars > 60)$
		2.B: $(CVC-Or-ISC) > 10(Cand Vars > 60)$
3	Which active constraint to choose for general branching?	3.A: <i>IVS-SIC</i>
		3.B: <i>CVS-SCC</i>
4	How to construct a 45-degree general disjunction that is approximately perpendicular to an equality constraint?	Switch sign of non-zero coefficients on even counter. If an odd number of non-zeros exist, then use <i>SC-var</i> to set one of the non-zero coefficients to zero in the disjunction function
5	Which branching direction to use for parallel general disjunctions?	5.A: <i>Sat-BC</i> (with offset) 5.B: <i>F-LP</i> (without offset)
6	Which branching direction to use for perpendicular general disjunctions?	<i>F-LP</i>

As can be seen from Table 10, there are two options for each of the second, third and fifth decisions; and a single option for each of the remaining decisions. There are eight combinations of methods altogether, which are compared against each other in this experiment in order to arrive at the best four algorithms for comparison against *Baseline_Cplex* in Section 8.

As can be seen in Figure 39, Figure 40, and Figure 41, the performances of the various algorithms are similar. Since all algorithms are run on the subset of MIP models which are solved by *Baseline_Cplex* in less than an hour, the robustness factor is mainly used for evaluating the performances of the different algorithms.

Figure 40 shows that the most robust combinations of options are "2.A, 3.B, 5.A", "2.B, 3.A, 5.A", and "2.B, 3.B, 5.A" as the performance profile graphs for these methods settle at the highest fraction of models solved with respect to the

other methods. Consequently, these methods are chosen as the top combinations of options, and are compared against *Baseline_Cplex* in the final experiment provided in Section 8.

The performances of "2.A, 3.B, 5.A" and "2.B, 3.B, 5.A" in the final experiment can be used to determine the best method of detecting when axis-parallel branching is performing poorly (either 2.A or 2.B). Similarly, the performances of "2.B, 3.A, 5.A" and "2.B, 3.B, 5.A" can be used to establish the best way of selecting an active constraint for general branching (either 3.A or 3.B). A fourth method is also included in the final comparison to assess the performances of the different branching-direction selection methods for parallel general disjunctions in the final set of experiments. Since the robustness of all other methods in Figure 40 are rather similar, "2.A, 3.B, 5.B" is picked as the fourth method as it solves the highest percentage of models over all methods before reaching a ratio to best of 2. Comparing the performances "2.A, 3.B, 5.B" and "2.A, 3.B, 5.A" in the final experiment, one can then determine which branching-direction selection method works best for parallel general disjunctions. The performances of "2.A, 3.B, 5.A", "2.B, 3.A, 5.A", "2.B, 3.B, 5.A", and "2.A, 3.B, 5.B" are compared against *Baseline_Cplex* in Section 8 of the thesis.

Figure 39: Experiment 5 performance profile on time for determining the best combinations of algorithms.

Figure 40: Experiment 5 performance profile on simplex iterations for determining the best combinations of algorithms.

Figure 41: Experiment 5 performance profile on node count for determining the best combinations of algorithms.

Section 7.2 showed that the performance of IVS-SIC is better than CVS-SCC in selecting an active constraint for general branching. However, the experiment in Section 7.2 did not enforce a lower limit on the number of candidate variables required before allowing the inclusion of a general disjunction. After imposing such limits on all methods compared here, one can note that three out of the four best methods chosen use CVS-SCC as their branching-constraint selection method, which makes CVS-SCC a conditionally effective method (i.e. effective when there are enough candidate variables at the current node in the B&B tree).

8. Comparison to the State of the Art

Comparing the best complete algorithms selected in Section 7 to the state-of-the-art Commercial MIP solver Cplex12.1 [Ilog 2009a, 2009b, 2009c] allows us to determine the real benefits of alternating axis-parallel and general disjunctions in order to find the first MIP-feasible solution as quickly as possible. The experiments presented in this section compare the top four algorithms as ranked in Section 7.5 (i.e. "2.A, 3.B, 5.A", "2.B, 3.A, 5.A", "2.B, 3.B, 5.A", and "2.A, 3.B, 5.B") against *Baseline_Cplex* over the subset of MIP problems that make up the testing set (See Appendix A.4 and A.5). All the algorithms are run using a time limit of eight hours. Section 8.1 compares the top four algorithms against *Baseline_Cplex* for all the hard MIP models listed in Appendix A.5. The use of general disjunctions is more often beneficial when the model is difficult enough that the benefit of adding general disjunctions outweighs the cost. As a result, comparing the performances of the top algorithms over all the hard MIP models allows for a better evaluation of the effectiveness of the algorithms. Section 8.2 compares the overall performances of the various algorithms against *Baseline_Cplex* for all the MIP models (i.e. easy and hard models) that are listed in Appendices A.4 and A.5, combined. Finally, Section 8.2 summarizes the comparison to the state of the art.

8.1. Comparing the Top Algorithms Against *Baseline_Cplex* Over the Hard MIPs

The performance profiles provided in Figure 42, Figure 43, and Figure 44 compare the performances of the top four algorithms against *Baseline_Cplex*

over the 111 hard MIP models listed in Appendix A.5. Out of the 111 models, all algorithms including *Baseline_Cplex* exceeded the time limit for 50% of the models, leaving 55 models that are solved by at least one of the algorithms before reaching the time limit. These models are listed in *italics* in Appendix A.5. The performance profiles presented here compare the performances of the different algorithms over these 55 models.

Figure 42, Figure 43, and Figure 44, show that all four general algorithms perform better than *Baseline_Cplex*. Before reaching a ratio to best of 5 in Figure 43, "2.A, 3.B, 5.B" and "2.B, 3.B, 5.A" complete 69% of the models at a faster rate than *Baseline_Cplex*, which only solves 60% of the models by that point. Similarly, the other two methods ("2.A, 3.B, 5.A" and "2.B, 3.A, 5.A") solve 68% of the models at a ratio to best of 5. All of the top algorithms are more robust than *Baseline_Cplex* as they solve higher fractions of models within the time limit in comparison to *Baseline_Cplex*. Table 11 summarizes the fractions of models solved within the time limit by each algorithm. In addition to robustness, "2.B, 3.A, 5.A" is the fastest 35% of the time (i.e. solves 35% of the models with a ratio to best of 1), whereas *Baseline_Cplex* is the fastest just 29% of the time.

Table 11: Percentages of models solved by each algorithm for all 55 hard MIP models.

	% of models solved within time limit by algorithm
2.A, 3.B, 5.A	69
2.A, 3.B, 5.B	71
2.B, 3.A, 5.A	69
2.B, 3.B, 5.A	71
<i>Baseline_Cplex</i>	67

The performance profile on node count, provided in Figure 44, shows that at a ratio to best of 2, "2.B, 3.B, 5.A" achieves the lowest node count for 69% of the

models, whereas "2.B, 3.A, 5.A" and *Baseline_Cplex* attain the lowest node count for only 60% and 59% of the models, respectively. At higher ratios to best, the performance of "2.B, 3.A, 5.A" picks up as it solves the highest fraction of models (71%). Furthermore, "2.B, 3.A, 5.A" attains the highest percentage of wins (38%) over all methods (including *Baseline_Cplex*) in the performance profiles.

Figure 42: Performance profile on time for comparing the top complete algorithms against *Baseline_Cplex* for the hard MIP models.

Figure 43: Performance profile on simplex iterations for comparing the top complete algorithms against *Baseline-Cplex* for the hard MIP models.

Figure 44: Performance profile on node count for comparing the top complete algorithms against *Baseline-Cplex* for the hard MIP models.

Given that the performances of all four general algorithms are relatively close, each algorithm is compared individually against *Baseline_Cplex* in Figure 45. The percentage of wins as well as the percentage of models solved at a ratio to best of 2 by each algorithm in comparison to *Baseline_Cplex* are summarized in Table 12.

Figure 45: Performance Profiles on simplex iterations comparing each of the top 4 methods against *Baseline_Cplex*.

Table 12 summarizes the percentages of models in which a particular algorithm achieves more than *Baseline_Cplex* with respect to the total number of wins and the robustness of the algorithm at a ratio to best of 2. A ratio to best of 2 is chosen as all the top algorithms are done before then. Using the results provided in Table 12, one can see that the best two methods in comparison to *Baseline_Cplex* are "2.A, 3.B, 5.A" and "2.B, 3.A, 5.A" as they attain the highest

percentages of wins or they solve the biggest fraction of models before reaching a ratio to best of 2. In addition to these two methods, "2.B, 3.B, 5.A" also solves the highest percentage of models before reaching a ratio to best of 2 in comparison to *Baseline_Cplex*.

Table 12: Summary of results for each of the top algorithms in comparison to *Baseline_Cplex*.

	% for (general algorithm) - % for <i>Baseline_Cplex</i>			
	2.A, 3.B, 5.A	2.A, 3.B, 5.B	2.B, 3.A, 5.A	2.B, 3.B, 5.A
% of wins	11	9	15	3
% of models solved at a ratio to best of 2	12	10	9	12

Comparing the performances of "2.B, 3.A, 5.A" and "2.B, 3.B, 5.A" against *Baseline_Cplex*, one can say that using measures based on integer variables (i.e. 3.A) for deciding on an active constraint for general branching generally leads to a fast and fairly robust algorithm, whereas using measures based on candidate variables (i.e. 3.B) leads to a slower but more robust algorithm. Similarly, if we compare the performances of "2.A, 3.B, 5.A" and "2.B, 3.B, 5.A" against *Baseline_Cplex*, one can say that using $(CVC-And-ISC)>3$ (i.e. 2.A) for deciding when to trigger the inclusion of a general disjunction leads to an algorithm that is generally faster than one that uses $(CVC-Or-ISC)>10$ (i.e. 2.B) for making that same decision. Since the performance of "2.A, 3.B, 5.A" in comparison to *Baseline_Cplex* is balanced with respect to speed and robustness, it is chosen as the recommended algorithm.

8.2. Overall Performances of Top Algorithms

Section 8.1 compared the performances of the top algorithms against *Baseline_Cplex* for the subset of MIP models which are categorized as "hard". In

order to determine if the top algorithms perform well for all levels of difficulty of MIP models, we compare their performances against *Baseline_Cplex* over all 112 models listed in Appendix A.4 (i.e. models which are solved in less than an hour by *Baseline_Cplex*) in addition to the 55 MIP models used in Section 8.1. A performance profile on simplex iterations comparing the performances of the top four algorithms with *Baseline_Cplex* over all 167 models from the testing set is provided in Figure 46.

The performance profile in Figure 46 shows that all algorithms are similar in performance to *Baseline_Cplex*. This is because the ratio of easy to hard models solved by the algorithms is quite high (112/55), and thus *Baseline_Cplex* has an advantage because the criterion for classifying a MIP model as easy is that Cplex solves it, and does so within an hour. Furthermore, 60% of the easy models listed in Appendix A.4 are solved by *Baseline_Cplex* in less than a minute, and thus the cost-benefit ratio of using general disjunctions on such models is fairly high.

At a ratio to best of 2, "2.B, 3.B, 5.A" solves 83% of the (167) models, while *Baseline_Cplex* solves 79% of the models at that ratio. "2.B, 3.B, 5.A" performs relatively better than the other three algorithms when compared to *Baseline_Cplex* over the easy MIP models as this algorithm on average generates the lowest number of general disjunctions per MIP model; hence the cost-benefit ratio of using general disjunctions is comparatively lower than the other algorithms. In order to observe significant improvements over *Baseline_Cplex* for the easy MIP models, one must use an algorithm that further limits the use of general disjunctions in such models.

Figure 46: Performance profile on simplex iterations for comparing the top complete algorithms against *Baseline_Cplex* for all types of MIP models.

Figure 47 demonstrates how alternating axis-parallel and general disjunctions can speed a MIP solution. It shows a typical example of the change in the number of candidate variables as we descend the B&B tree for *Baseline_Cplex* in comparison to "2.B, 3.B, 5.A" for a problem from the easy test set (*Satellites1-25*). "2.B, 3.B, 5.A" triggers the inclusion of four general disjunctions (including one at the root node) during the MIP solution. As shown in Figure 47, *Baseline_Cplex* requires more nodes to solve the problem and stalls frequently during the solution process, and thus takes 12 minutes to attain a feasible solution. In contrast, "2.B, 3.B, 5.A" requires less than a minute to solve the problem by recognizing when the MIP solution is stalling, which triggers the inclusion of a "45-degree" general disjunction.

Figure 47: Comparing the rate of decrease in number of candidate variables for *Baseline_Cplex* Vs "2.B, 3.B, 5.A" for *Satellites1-25*.

Figure 48 looks more closely at the rate of decrease in number of candidate variables and infeasibility sum when "2.B, 3.B, 5.A" is applied to this model. Aside from the root node, a vertical green line highlights the points at which a general disjunction was triggered. A general disjunction is triggered if either the number of candidate variables or the infeasibility sum increases or stays the same more than 10 times in a row (i.e. method 2.B).

Figure 48: Observing the rate of decrease in number of candidate variables and infeasibility sum at different nodes for "2.B, 3.B, 5.A" for *Satellites1-25*.

As mentioned in Section 6.4, the number of backtracks is another measure of the quality of a B&B algorithm. On average, the number of backtracks performed by any of the top algorithms is less than (or equal to) those performed by *Baseline_Cplex* in 60% of the models listed in Appendices A.4 and A.5. Exact percentages for each of the top algorithms are provided in Table 13.

Table 13: Comparing the number of backtracks performed by each of the top algorithms versus *Baseline_Cplex*.

	% of models in which the number of backtracks is less than (or equal to) <i>Baseline_Cplex</i> 's backtracks
2.A, 3.B, 5.A	59
2.A, 3.B, 5.B	50
2.B, 3.A, 5.A	68
2.B, 3.B, 5.A	64

The average optimality gap (which is used to measure the quality of the first MIP-feasible solution) for each of top algorithms, is compared against *Baseline_Cplex* for all problems in the testing set in which both the compared

algorithm and *Baseline_Cplex* did not exceed the time limit. On average, the optimality gap of any of the top algorithms is better than (or equal to) that of *Baseline_Cplex* in 55-60% of the models, as shown in Table 14.

To sum up, the top algorithms are generally faster, with fewer backtracks and lower optimality gaps than *Baseline_Cplex*.

Table 14: Comparing the optimality gap of each of the top algorithms versus *Baseline_Cplex*.

	% of models in which the optimality gap is better than (or equal to) <i>Baseline_Cplex</i> 's optimality gap
2.A, 3.B, 5.A	59
2.A, 3.B, 5.B	55
2.B, 3.A, 5.A	55
2.B, 3.B, 5.A	60

Preliminary experiments not reported here, but similar to those presented in Sections 7 and 8 of the thesis, were also conducted using the open-source GLPK 4.38 MIP solver [Makhorin 2009]. Several general disjunction algorithms were compared with GLPK's default B&B algorithm with parameters set to encourage early integer feasibility, and with Method A [Patel and Chinneck, 2007] for choosing the branching variable, one of the fastest algorithms for reaching integer feasibility quickly. Significant speed improvements over both of these methods were observed. The qualities of the solutions returned using general disjunctions were also better than GLPK's default B&B algorithm on average 60% of the time. These observations support the results reported here concerning the value of using "45-degree" general disjunctions for reaching MIP-feasibility quickly.

9. Conclusions and Future Work

9.1. Conclusions

The following is a summary of inferences made by this research:

- Using a larger set of disjunctions in the MIP solution can have a positive significant impact on the speed of the solution.
- Inserting "45-degree" disjunctions of the types developed in this thesis speeds the MIP solution by forcing multiple candidate variables to change simultaneously.
- Triggering the inclusion of general disjunctions is cost-effective if the solution progress is stalling while using axis-parallel branching and if there are enough candidate variables at the current node in the tree.
- It is generally good to include a general disjunction after the LP-relaxation of the root node, but only if there are sufficient candidate variables.
- The most effective branching-constraint selection schemes (in those that were studied in this thesis) are those that use the number of integer or candidate variables as well as the sum of integer or candidate variable absolute coefficients to choose the active constraint that has the greatest impact on the MIP solution.
- The best way (among the alternatives considered in this thesis) of creating an approximately perpendicular general disjunction is to alternate the signs of non-zero coefficients on even count in the disjunction function. If the number of non-zero coefficients in the branching constraint is zero,

then it is best to set the coefficient of the variable having the smallest coefficient to zero in the disjunction function, as this variable has the least impact on the LP solution.

- The best branching direction for the approximately parallel general disjunctions developed in this thesis is either in the satisfying direction of the branching constraint or in the direction that is farther from the LP-relaxation optimum point. These two branching directions tend to force the greatest number of candidate variables to change.
- The best branching direction for the approximately perpendicular general disjunctions developed in this thesis is branching farther from the LP-relaxation optimum point, as this also forces the greatest number of candidate variables to change.

9.2. Summary of Contributions

The following is a summary of contributions to research made by this thesis:

- Introduced the idea of alternating between axis-parallel and general disjunctions in the same MIP solution in order to attain the first MIP-feasible solution as quickly as possible.
- Established new ways of recognizing when a MIP solver is stalling; monitoring changes in the number of candidate variables along with the infeasibility sum in the B&B tree can help determine when one type of branching method is performing poorly.
- Developed rules for deciding if it is cost-effective to trigger the inclusion of a general disjunction in the MIP solution (even at the root node).

- Introduced ways of constructing useful "45-degree" general disjunctions that are approximately parallel or perpendicular to the active constraint that has the most impact on achieving MIP-feasibility quickly.
- Provided further evidence of the importance of branching in the direction that forces the greatest number of candidate variables to change simultaneously; either by branching in the satisfying direction of the inequality branching constraint or in the direction that is farther from the LP-relaxation optimum point.
- Developed four effective methods ("2.A, 3.B, 5.A", "2.B, 3.A, 5.A", "2.B, 3.B, 5.A", and "2.A, 3.B, 5.B") that show improvements over a state-of-the-art commercial MIP solver (Cplex).

9.3. Future Work

The following is a list of future research that could be conducted to expand on the contributions made by this thesis:

- Instead of choosing only one constraint from the set of active constraints, one can develop an artificial constraint that is based on all active constraints (or all equally-ranked active constraints). One can then create a general disjunction that is approximately parallel or perpendicular to that constraint. On a similar note, one can create an artificial constraint that is based on those active constraints that do not overlap (i.e. do not have candidate variables in common). This would then help create a general disjunction that impacts more candidate variables at once.

- Measure the amount of improvement realized by a general disjunction and determine if such improvement is worth the cost of creating the general disjunction. If one determines that a great amount of improvement is achieved by the general disjunction, then perhaps it is worth the effort to perform another general disjunction later. If the cost of creating a general disjunction outweighs its benefit, one can stop using general disjunctions and go back to using axis-parallel disjunctions. This idea can be extended by keeping track of the performance gains achieved by branching on particular constraints and then use such measures to break the ties between equally-ranked active constraints.
- Given that the initial disjunction used can greatly impact the speed of the B&B algorithm, various properties of the MIP problem at the root node, such as the type of problem (MIP, BIP, or IP), initial percentage of candidate variables, initial percentage of active constraints, and the type of constraints in the problem can help decide on the type of disjunction to use at the root node.
- In addition to using parallel general disjunctions for inequality branching constraints, one can also explore the idea of using perpendicular general disjunctions for such constraints. If the coefficients of the branching constraint are the same as the parallel general disjunction (i.e. in "multiple choice" constraints) then one could create a general disjunction that is approximately

perpendicular to that constraint instead of offsetting the right-hand side value of the parallel disjunction by one unit.

- Discover new ways of effectively monitoring the progress of the MIP solution, such as the growth rate in number of active nodes while using axis-parallel disjunctions in order to trigger the inclusion of a general disjunction if axis-parallel branching is performing poorly.
- Parallelize the algorithms implemented here to take advantage of Cplex's ability to parallelize the B&B algorithm on multiple threads. All the algorithms can be directly used in parallel B&B.

10. References

- Achterberg, T., Koch, T. and Martin, A. Branching rules revisited. *Operations Research Letters* **33**, 42-54 (2004).
- Benichou, M., Gauthier, J.M., Girodet, P., Hentges, G., Ribiere, G., and Vincent, O. Experiments in mixed-integer linear programming. *Mathematical Programming* **1**, 76-94 (1971).
- Chen, D.-S., Batson, R.G. and Dang, Y. *Applied Integer Programming: Modeling and Solution*. John Wiley & Sons, Inc., Hoboken, NJ, USA (2009).
- Chinneck, J.W. *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*. International Series in Operations Research and Management Sciences 118, Springer, Boston, MA (2008).
- Chlamtac, E. and Tulsiani, M. *Convex relaxations and integrality gaps*. Handbook on Semidefinite, Cone and Polynomial Optimization, Springer (2011).
- Cornuéjols, G., Liberti, L. and Nannicini, G. Improved Strategies for Branching on General Disjunctions. *Math. Program.* **130**, no. 2, 225-247 (2011).
- Dolan, E.D., and Moré, J. Benchmarking Optimization Software with Performance Profiles. *Mathematical Programming. Series A*, **91**, 201-213 (2002).
- Fischetti, M., Glover, F. and Lodi, A. The Feasibility Pump. *Math. Program.* **104**, 91-104 (2005).
- ILOG Corporation (2009a). Cplex 12.1 Parameters Reference Manual.
- ILOG Corporation (2009b). Cplex 12.1 C API Reference Manual.
- ILOG Corporation (2009c). Cplex 12.1 User's Manual.
- INFORMS Computer Society, Mathematical Programming Glossary, <http://glossary.computing.society.informs.org/index.php?page=O.html>, [Accessed December 2011].
- Karamanov, M., Cornuéjols, G., Branching on General Disjunctions. *Math. Program.* **128**, no. 1-2, 403-436 (2011).
- Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., Lodi, A., Mittelman, H., Ralphs, T., Salvagnin, D., Steffy, D.E., Wolter, K. "MIP LIB 2010", *Mathematical Programming Computation*, **3**, no. 2, 103-163 (2011).

Mahajan, A. and Ralphs, T. On the Complexity of Selecting Disjunctions in Integer Programming. *SIAM J. on Optimization* **20**, 2181–2198 (2010).

Mahajan, A. and Ralphs, T.K. Experiments with Branching using General Disjunctions. *Operations Research and Cyber-Infrastructure* **47**, 101-118 (2009).

Makhorin, A., “GNU Linear Programming Kit Reference Manual Version 4.38”, 2009, <http://www.gnu.org/software/glpk/>, [Accessed December 2011].

Microsoft Corporation (2007). Microsoft .Net Framework.

Owen, J.H. and Mehrotra, S. Experimental Results on using General Disjunctions in Branch-and-Bound for General-Integer Linear Programs. *Comput. Optim. Appl.* **20**, 159–170 (2001).

Meyer, R.R., D'Souza, W.D., Ferris, M.C., and Thomadsen, B.R. MIP Models and BB Strategies in Brachytherapy Treatment Optimization. *Journal of Global Optimization* **25**, 23–42 (2003).

Patel, J. and Chinneck, J.W. Active-Constraint Variable Ordering for Faster Feasibility of Mixed Integer Linear Programs. *Math. Program.* **110**, 445-474 (2007).

Pryor, J. and Chinneck, J.W. Faster Integer-Feasibility in Mixed-Integer Linear Programs by Branching to Force Change. *Computers & Operations Research* **38**, 1143-1152 (2011).

Wojtaszek, D.T. and Chinneck, J.W. Faster MIP Solutions via New Node Selection Rules. *Computers & Operations Research* **37**, 1544-1556 (2010).

A.1 MIPLIB 2010 Subset of Equality-Containing MIP Models

The table below lists the 100 randomly-chosen easy models used in training experiments requiring equality-containing models. Statistics on the numbers and types of variables and constraints in each model are listed. Average values are given at the end of the table.

MIP Model	Number of Constraints	Number of Variables	Number of Binary Vars.	Number of Integer Vars.	Number of Real Vars.
janos-us-DDM.mps	760	2184	0	84	2100
b2c1s1.mps	3904	3872	288	0	3584
g200x740i.mps	940	1480	740	0	740
neos-1396125.mps	1494	1161	129	0	1032
neos-506428.mps	129925	42981	42981	0	0
neos-693347.mps	3192	1576	1405	0	171
neos-799711.mps	59218	41998	910	0	41088
beasleyC3.mps	1750	2500	1250	0	1250
dg012142.mps	6310	2080	640	0	1440
neos-826841.mps	2354	5516	3488	0	2028
neos-1112782.mps	2115	4140	2070	0	2070
neos-777800.mps	479	6400	6400	0	0
dano3mip.mps	3202	13873	552	0	13321
k16x240.mps	256	480	240	0	240
lectsched-4-obj.mps	14163	7901	7665	236	0
neos-1605075.mps	3467	4173	3633	0	540
neos-935769.mps	6741	9799	7020	0	2779
acc-tight5.mps	3052	1339	1339	0	0
app1-2.mps	53467	26871	13300	0	13571
gmu-35-40.mps	424	1205	1200	0	5
neos-941262.mps	6703	9480	6710	0	2770
neos-916792.mps	1909	1474	717	0	757
berlin_5_8_0.mps	1532	1083	794	0	289

bienst2.mps	576	505	35	0	470
map18.mps	328818	164547	146	0	164401
ger50_17_trans.mps	499	22414	0	18062	4352
d20200.mps	1502	4000	3181	819	0
mcsched.mps	2107	1747	1731	14	2
neos-506422.mps	6811	2527	63	0	2464
danoint.mps	664	521	56	0	465
neos-826812.mps	6844	15864	10350	0	5514
50v-10.mps	233	2013	1464	183	366
neos-847302.mps	609	737	729	0	8
neos-824661.mps	18804	45390	15640	0	29750
neos-911880.mps	83	888	840	0	48
co-100.mps	2187	48417	48417	0	0
gmut-75-50.mps	2565	68865	68859	0	6
dolom1.mps	1803	11612	9720	0	1892
map20.mps	328818	164547	146	0	164401
dfn-gwin-UUM.mps	158	938	0	90	848
eilA101.2.mps	100	65832	65832	0	0
neos-1109824.mps	28979	1520	1520	0	0
buildingenergy.mps	277594	154978	0	26287	128691
n3700.mps	5150	10000	5000	0	5000
go19.mps	441	441	441	0	0
neos-555424.mps	2676	3815	3800	15	0
neos-1337307.mps	5687	2840	2840	0	0
nag.mps	5840	2884	1350	35	1499
acc-tight4.mps	3285	1620	1620	0	0
csched010.mps	351	1758	1457	0	301
30n20b8.mps	576	18380	11036	7344	0
neos-476283.mps	10015	11915	5588	0	6327
bg512142.mps	1307	792	240	0	552
csched008.mps	351	1536	1284	0	252
lotsize.mps	1920	2985	1195	0	1790
acc-tight6.mps	3047	1335	1335	0	0
neos-1620770.mps	9296	792	792	0	0
map14.mps	328818	164547	146	0	164401

neos-824695.mps	9576	23970	8500	0	15470
neos-1225589.mps	675	1300	650	0	650
n4-3.mps	1236	3596	0	174	3422
neos-941313.mps	13189	167910	167910	0	0
germany50-DBM.mps	2526	8189	0	88	8101
neos-942830.mps	803	882	834	0	48
neos-520729.mps	31178	91149	30708	0	60441
map06.mps	328818	164547	146	0	164401
n9-3.mps	2364	7644	0	252	7392
glass4.mps	396	322	302	0	20
msc98-ip.mps	15850	21143	20237	53	853
markshare_5_0.mps	5	45	40	0	5
binkar10_1.mps	1026	2298	170	0	2128
mc11.mps	1920	3040	1520	0	1520
gmu-35-50.mps	435	1919	1914	0	5
n3-3.mps	2425	9028	0	366	8662
mkc.mps	3411	5325	5323	0	2
blp-ar98.mps	1128	16021	15806	0	215
blp-ic97.mps	923	9845	9753	0	92
neos-826694.mps	6904	16410	16290	0	120
neos-1112787.mps	1680	3280	1640	0	1640
neos-1605061.mps	3474	4111	3570	0	541
leo1.mps	593	6731	6730	0	1
n370a.mps	5150	10000	5000	0	5000
dc1c.mps	1649	10039	8380	0	1659
eilB101.mps	100	2818	2818	0	0
d10200.mps	947	2000	733	1267	0
biella1.mps	1203	7328	6110	0	1218
dc1l.mps	1653	37297	35638	0	1659
atlanta-ip.mps	21732	48738	46667	106	1965
mzzv11.mps	9499	10240	9989	251	0
map10.mps	328818	164547	146	0	164401
leo2.mps	593	11100	11099	0	1
maxgasflow.mps	7160	7437	2456	0	4981
gmut-77-40.mps	2554	24338	24332	0	6
eil33.2.mps	32	4516	4516	0	0

n3705.mps	5150	10000	5000	0	5000
csched007.mps	351	1758	1457	0	301
harp2.mps	112	2993	2993	0	0
neos- 1440225.mps	330	1285	1285	0	0
air04.mps	823	8904	8904	0	0
neos- 1601936.mps	3131	4446	3906	0	540
AVERAGE	25173.93	21355.57	8437.96	557.26	12360.35

A.2 MIPLIB 2010 Subset of Inequality-Containing MIP

Models

The table below lists the 100 randomly-chosen easy models used in training experiments requiring inequality-containing models. Statistics on the numbers and types of variables and constraints in each model are listed. Average values are given at the end of the table. The subset of 72 models in which only parallel general disjunctions were triggered by all algorithms in Experiment 4.A (Section 7.4.1) are *italicized*.

MIP Model	Number of Constraints	Number of Variables	Number of Binary Vars.	Number of Integer Vars.	Number of Real Vars.
biella1.mps	1203	7328	6110	0	1218
<i>neos-1616732.mps</i>	1999	200	200	0	0
<i>cov1075.mps</i>	637	120	120	0	0
<i>dc1l.mps</i>	1653	37297	35638	0	1659
<i>berlin_5_8_0.mps</i>	1532	1083	794	0	289
<i>mzzv11.mps</i>	9499	10240	9989	251	0
<i>germany50-DBM.mps</i>	2526	8189	0	88	8101
<i>gmut-77-40.mps</i>	2554	24338	24332	0	6
<i>gmu-35-50.mps</i>	435	1919	1914	0	5
<i>neos-1171737.mps</i>	4179	2340	1170	0	1170
<i>ic97_potential.mps</i>	1046	728	450	73	205
<i>binkar10_1.mps</i>	1026	2298	170	0	2128
<i>maxgasflow.mps</i>	7160	7437	2456	0	4981
<i>macrophage.mps</i>	3164	2260	2260	0	0
<i>danoimt.mps</i>	664	521	56	0	465
<i>d20200.mps</i>	1502	4000	3181	819	0
<i>acc-tight6.mps</i>	3047	1335	1335	0	0
<i>liu.mps</i>	2178	1156	1089	0	67
<i>map14.mps</i>	328818	164547	146	0	164401

<i>neos-1442119.mps</i>	1524	728	364	0	364
<i>n3700.mps</i>	5150	10000	5000	0	5000
<i>neos-1605061.mps</i>	3474	4111	3570	0	541
<i>harp2.mps</i>	112	2993	2993	0	0
<i>dc1c.mps</i>	1649	10039	8380	0	1659
<i>blp-ar98.mps</i>	1128	16021	15806	0	215
<i>bg512142.mps</i>	1307	792	240	0	552
<i>map06.mps</i>	328818	164547	146	0	164401
<i>blp-ic97.mps</i>	923	9845	9753	0	92
<i>bienst2.mps</i>	576	505	35	0	470
<i>30n20b8.mps</i>	576	18380	11036	7344	0
<i>glass4.mps</i>	396	322	302	0	20
<i>mik.250-1-100.1.mps</i>	151	251	100	150	1
<i>csched008.mps</i>	351	1536	1284	0	252
<i>leo2.mps</i>	593	11100	11099	0	1
<i>buildingenergy.mps</i>	277594	154978	0	26287	128691
<i>n3705.mps</i>	5150	10000	5000	0	5000
<i>mc11.mps</i>	1920	3040	1520	0	1520
<i>lectsched-4-obj.mps</i>	14163	7901	7665	236	0
<i>neos-1337307.mps</i>	5687	2840	2840	0	0
<i>mine-166-5.mps</i>	8429	830	830	0	0
<i>n370a.mps</i>	5150	10000	5000	0	5000
<i>neos-1442657.mps</i>	1310	624	312	0	312
<i>dano3mip.mps</i>	3202	13873	552	0	13321
<i>n3seq24.mps</i>	6044	119856	119856	0	0
<i>neos-1396125.mps</i>	1494	1161	129	0	1032
<i>neos-1109824.mps</i>	28979	1520	1520	0	0
<i>iis-pima-cov.mps</i>	7201	768	768	0	0
<i>neos-1112782.mps</i>	2115	4140	2070	0	2070
<i>iis-bupa-cov.mps</i>	4803	345	345	0	0
<i>dfn-gwin-UUM.mps</i>	158	938	0	90	848
<i>lotsize.mps</i>	1920	2985	1195	0	1790
<i>n3-3.mps</i>	2425	9028	0	366	8662
<i>b2c1s1.mps</i>	3904	3872	288	0	3584

co-100.mps	2187	48417	48417	0	0
n9-3.mps	2364	7644	0	252	7392
ex1010-pi.mps	1468	25200	25200	0	0
neos-1426635.mps	796	520	260	0	260
acc-tight5.mps	3052	1339	1339	0	0
map20.mps	328818	164547	146	0	164401
acc-tight4.mps	3285	1620	1620	0	0
beasleyC3.mps	1750	2500	1250	0	1250
app1-2.mps	53467	26871	13300	0	13571
mcsched.mps	2107	1747	1731	14	2
mine-90-10.mps	6270	900	900	0	0
n3div36.mps	4484	22120	22120	0	0
neos-1436709.mps	1417	676	338	0	338
neos-1440460.mps	989	468	234	0	234
map10.mps	328818	164547	146	0	164401
gmut-75-50.mps	2565	68865	68859	0	6
mkc.mps	3411	5325	5323	0	2
dg012142.mps	6310	2080	640	0	1440
atlanta-ip.mps	21732	48738	46667	106	1965
neos-1605075.mps	3467	4173	3633	0	540
neos-1601936.mps	3131	4446	3906	0	540
dolom1.mps	1803	11612	9720	0	1892
iis-100-0-cov.mps	3831	100	100	0	0
janos-us-DDM.mps	760	2184	0	84	2100
leo1.mps	593	6731	6730	0	1
g200x740i.mps	940	1480	740	0	740
k16x240.mps	256	480	240	0	240
neos-1112787.mps	1680	3280	1640	0	1640
m100n500k4r1.mps	100	500	500	0	0
core2536-691.mps	2539	15293	15284	0	9
gmu-35-40.mps	424	1205	1200	0	5
nag.mps	5840	2884	1350	35	1499
csched010.mps	351	1758	1457	0	301

<i>csched007.mps</i>	351	1758	1457	0	301
<i>go19.mps</i>	441	441	441	0	0
<i>msc98-ip.mps</i>	15850	21143	20237	53	853
<i>30_70_4.5_0.9_5_100.mps</i>	12526	10976	10975	0	1
<i>methanosarcina.mps</i>	14604	7930	7930	0	0
<i>neos-1426662.mps</i>	1914	832	416	0	416
<i>neos-1171692.mps</i>	4239	1638	819	0	819
<i>ger50_17_trans.mps</i>	499	22414	0	18062	4352
<i>n4-3.mps</i>	1236	3596	0	174	3422
<i>50v-10.mps</i>	233	2013	1464	183	366
<i>d10200.mps</i>	947	2000	733	1267	0
<i>neos-1311124.mps</i>	1643	1092	546	0	546
<i>map18.mps</i>	328818	164547	146	0	164401
<i>neos-1225589.mps</i>	675	1300	650	0	650
AVERAGE	22881.79	17691.35	6422.12	559.34	10709.89

A.3 MIPLIB 2010 Subset of Equality and Inequality-Containing MIP Models

The table below lists the 100 randomly-chosen easy models used in training experiments requiring equality and inequality-containing models. Statistics on the numbers and types of variables and constraints in each model are listed.

Average values are given at the end of the table.

MIP Model	Number of Constraints	Number of Variables	Number of Binary Vars.	Number of Integer Vars.	Number of Real Vars.
neos-1440225.mps	330	1285	1285	0	0
buildingenergy.mps	277594	154978	0	26287	128691
neos-1112782.mps	2115	4140	2070	0	2070
csched008.mps	351	1536	1284	0	252
n370a.mps	5150	10000	5000	0	5000
n3seq24.mps	6044	119856	119856	0	0
gmu-35-40.mps	424	1205	1200	0	5
map06.mps	328818	164547	146	0	164401
cov1075.mps	637	120	120	0	0
csched010.mps	351	1758	1457	0	301
neos-1311124.mps	1643	1092	546	0	546
neos-1109824.mps	28979	1520	1520	0	0
core2536-691.mps	2539	15293	15284	0	9
germany50-DBM.mps	2526	8189	0	88	8101
maxgasflow.mps	7160	7437	2456	0	4981
dfn-gwin-UUM.mps	158	938	0	90	848
app1-2.mps	53467	26871	13300	0	13571
acc-tight4.mps	3285	1620	1620	0	0
glass4.mps	396	322	302	0	20
bienst2.mps	576	505	35	0	470
map20.mps	328818	164547	146	0	164401
dg012142.mps	6310	2080	640	0	1440
biella1.mps	1203	7328	6110	0	1218

dc1c.mps	1649	10039	8380	0	1659
30n20b8.mps	576	18380	11036	7344	0
neos-1396125.mps	1494	1161	129	0	1032
acc-tight5.mps	3052	1339	1339	0	0
map10.mps	328818	164547	146	0	164401
gmut-75-50.mps	2565	68865	68859	0	6
d20200.mps	1502	4000	3181	819	0
leo1.mps	593	6731	6730	0	1
mik.250-1-100.1.mps	151	251	100	150	1
neos-1112787.mps	1680	3280	1640	0	1640
gmut-77-40.mps	2554	24338	24332	0	6
n9-3.mps	2364	7644	0	252	7392
n3705.mps	5150	10000	5000	0	5000
n3div36.mps	4484	22120	22120	0	0
neos-1171737.mps	4179	2340	1170	0	1170
mkc.mps	3411	5325	5323	0	2
msc98-ip.mps	15850	21143	20237	53	853
neos-1426662.mps	1914	832	416	0	416
acc-tight6.mps	3047	1335	1335	0	0
30_70_4.5_0.95_100.mps	12526	10976	10975	0	1
iis-pima-cov.mps	7201	768	768	0	0
atlanta-ip.mps	21732	48738	46667	106	1965
danoint.mps	664	521	56	0	465
b2c1s1.mps	3904	3872	288	0	3584
eilA101.2.mps	100	65832	65832	0	0
leo2.mps	593	11100	11099	0	1
dc1l.mps	1653	37297	35638	0	1659
50v-10.mps	233	2013	1464	183	366
mcsched.mps	2107	1747	1731	14	2
macrophage.mps	3164	2260	2260	0	0
gmu-35-50.mps	435	1919	1914	0	5
ger50_17_trans.mps	499	22414	0	18062	4352
dano3mip.mps	3202	13873	552	0	13321
eilB101.mps	100	2818	2818	0	0
blp-ic97.mps	923	9845	9753	0	92
m100n500k4r1.mps	100	500	500	0	0

harp2.mps	112	2993	2993	0	0
neos-1171692.mps	4239	1638	819	0	819
n3-3.mps	2425	9028	0	366	8662
methanosarcina.mps	14604	7930	7930	0	0
d10200.mps	947	2000	733	1267	0
iis-bupa-cov.mps	4803	345	345	0	0
mc11.mps	1920	3040	1520	0	1520
go19.mps	441	441	441	0	0
ic97_potential.mps	1046	728	450	73	205
dolom1.mps	1803	11612	9720	0	1892
neos-1426635.mps	796	520	260	0	260
bg512142.mps	1307	792	240	0	552
n3700.mps	5150	10000	5000	0	5000
mzzv11.mps	9499	10240	9989	251	0
neos-1225589.mps	675	1300	650	0	650
mine-166-5.mps	8429	830	830	0	0
mine-90-10.mps	6270	900	900	0	0
air04.mps	823	8904	8904	0	0
csched007.mps	351	1758	1457	0	301
neos-1440460.mps	989	468	234	0	234
ex1010-pi.mps	1468	25200	25200	0	0
neos-1337307.mps	5687	2840	2840	0	0
janos-us-DDM.mps	760	2184	0	84	2100
map18.mps	328818	164547	146	0	164401
markshare_5_0.mps	5	45	40	0	5
binkar10_1.mps	1026	2298	170	0	2128
co-100.mps	2187	48417	48417	0	0
neos-1436709.mps	1417	676	338	0	338
beasleyC3.mps	1750	2500	1250	0	1250
iis-100-0-cov.mps	3831	100	100	0	0
lotsize.mps	1920	2985	1195	0	1790
nag.mps	5840	2884	1350	35	1499
blp-ar98.mps	1128	16021	15806	0	215
lectsched-4-obj.mps	14163	7901	7665	236	0
liu.mps	2178	1156	1089	0	67

g200x740i.mps	940	1480	740	0	740
map14.mps	328818	164547	146	0	164401
n4-3.mps	1236	3596	0	174	3422
berlin_5_8_0.mps	1532	1083	794	0	289
eil33.2.mps	32	4516	4516	0	0
k16x240.mps	256	480	240	0	240
AVERAGE	22746.64	18382.53	7136.22	559.34	10686.97

A.4 MIPLIB 2010 Subset of Easy MIP Models Used for

Testing

The table below lists the 112 easy models used in testing experiments requiring equality and inequality-containing models. Statistics on the numbers and types of variables and constraints in each model are listed. Average values are given at the end of the table.

MIP Model	Number of Constraints	Number of Variables	Number of Binary Vars.	Number of Integer Vars.	Number of Real Vars.
ash608gpia-3col.mps	24748	3651	3651	0	0
enlight9.mps	81	162	81	81	0
enlight13.mps	169	338	169	169	0
neos13.mps	20852	1827	1815	0	12
neos15.mps	552	792	160	0	632
neos16.mps	1018	377	336	41	0
neos18.mps	11402	3312	3312	0	0
neos6.mps	1036	8786	8340	0	446
neos788725.mps	433	352	352	0	0
neos-885086.mps	11574	4860	2430	0	2430
neos-933638.mps	13658	32417	28637	0	3780
neos-933966.mps	12047	31762	27982	0	3780
neos-934278.mps	11495	23123	19955	0	3168
neos-948126.mps	7271	9551	6965	0	2586
neos-957389.mps	5115	6036	6036	0	0
newdano.mps	576	505	56	0	449
nobel-eu-DBE.mps	879	3771	1639	0	2132
noswot.mps	182	128	75	25	28
npmv07.mps	76342	220686	1880	0	218806
ns1111636.mps	13895	360822	13200	0	347622
ns1158817.mps	68455	1804022	66022	0	1738000
ns1606230.mps	3503	4173	3633	0	540
ns1686196.mps	4055	2738	2738	0	0

ns1688347.mps	4191	2685	2685	0	0
ns1702808.mps	1474	804	666	0	138
ns1766074.mps	182	100	0	90	10
ns1830653.mps	2932	1629	1458	0	171
ns1856153.mps	35407	11998	11956	0	42
ns2017839.mps	54510	55224	12	0	55212
ns2081729.mps	1190	661	600	0	61
ns2124243.mps	139280	156083	16447	0	139636
ns4-pr3.mps	2210	8601	0	61	8540
ns4-pr9.mps	2220	7350	0	42	7308
ns894244.mps	12129	21856	21856	0	0
ns930473.mps	23240	11328	11176	0	152
nu120-pr3.mps	2210	8601	8540	61	0
nu60-pr9.mps	2220	7350	7308	42	0
ofi.mps	422587	420434	18632	11073	390729
opm2-z7-s2.mps	31798	2023	2023	0	0
p100x588b.mps	688	1176	588	0	588
p6b.mps	5852	462	462	0	0
p80x400b.mps	480	800	400	0	400
pg.mps	125	2700	100	0	2600
pg5_34.mps	225	2600	100	0	2500
pigeon-10.mps	931	490	400	0	90
pigeon-11.mps	1123	572	473	0	99
pigeon-12.mps	1333	660	552	0	108
pigeon-13.mps	1561	754	637	0	117
pigeon-19.mps	3307	1444	1273	0	171
protfold.mps	2112	1835	1835	0	0
pw-myciel4.mps	8164	1059	1058	1	0
qiu.mps	1192	840	48	0	792
queens-30.mps	960	900	900	0	0
r80x800.mps	880	1600	800	0	800
rail507.mps	509	63019	63009	0	10
ramos3.mps	2187	2187	2187	0	0
ran14x18.disj-8.mps	447	504	252	0	252
ran14x18.mps	284	504	252	0	252
ran16x16.mps	288	512	256	0	256
reblock166.mps	17024	1660	1660	0	0
reblock354.mps	19906	3540	3540	0	0
reblock420.mps	62800	4200	4200	0	0
reblock67.mps	2523	670	670	0	0

rmatr100-p10.mps	7260	7359	100	0	7259
rmatr100-p5.mps	8685	8784	100	0	8684
rmatr200-p10.mps	35055	35254	200	0	35054
rmatr200-p20.mps	29406	29605	200	0	29405
rmatr200-p5.mps	37617	37816	200	0	37616
rmine6.mps	7078	1096	1096	0	0
rocll-4-11.mps	21738	9234	9086	0	148
rococoB10-011000.mps	1667	4456	4320	136	0
rococoC10-001000.mps	1293	3117	2993	124	0
rococoC11-011100.mps	2367	6491	6325	166	0
roll3000.mps	2295	1166	246	492	428
rvb-sub.mps	225	33765	33763	0	2
satellites1-25.mps	5996	9013	8509	0	504
sct1.mps	12154	22886	9044	1268	12574
sct5.mps	13304	37265	20702	2302	14261
set3-10.mps	3747	4019	1424	0	2595
set3-15.mps	3747	4019	1424	0	2595
set3-20.mps	3747	4019	1424	0	2595
seymour.disj-10.mps	5108	1209	1209	0	0
seymour.mps	4944	1372	1372	0	0
siena1.mps	2220	13741	11775	0	1966
sing2.mps	28891	31630	23377	0	8253
sing245.mps	143161	235146	220692	0	14454
sp97ar.mps	1761	14101	14101	0	0
sp98ic.mps	825	10894	10894	0	0
sp98ir.mps	1531	1680	871	809	0
sts405.mps	27270	405	405	0	0
sts729.mps	88452	729	729	0	0
swath.mps	884	6805	6724	0	81
t1717.mps	551	73885	73885	0	0
t1722.mps	338	36630	36630	0	0
tanglegram1.mps	68342	34759	34759	0	0
tanglegram2.mps	8980	4714	4714	0	0
timtab1.mps	171	397	64	107	226
toll-like.mps	4408	2883	2883	0	0

transportmomen t.mps	9616	9685	2456	0	7229
triptim1.mps	15706	30055	20451	9597	7
tw-myciel4.mps	8146	760	759	1	0
uc-case11.mps	51438	34134	3898	302	29934
uct- subprob.mps	1973	2256	379	0	1877
umts.mps	4465	2947	2802	72	73
unitcal_7.mps	48939	25755	2856	0	22899
usAbbrev.8.25_7 0.mps	3291	2312	1681	0	631
van.mps	27331	12481	192	0	12289
vpphard.mps	47280	51471	51471	0	0
vpphard2.mps	198450	199999	199999	0	0
wachplan.mps	1553	3361	3360	1	0
wnq-n100- mw99-14.mps	656900	10000	10000	0	0
zib54-UUE.mps	1809	5150	81	0	5069
AVERAGE	25215.48	39788.54	11027.68	241.63	28519.22

A.5 MIPLIB 2010 Subset of Hard MIP Models Used for Testing

The table below lists the 111 hard models used in testing experiments requiring equality and inequality-containing models. Statistics on the numbers and types of variables and constraints in each model are listed. Average values are given at the end of the table. Models in which at least one algorithm was able to solve within the time limit are listed in *italics*.

MIP Model	Number of Constraints	Number of Variables	Number of Binary Vars.	Number of Integer Vars.	Number of Real Vars.
<i>a1c1s1.mps</i>	3312	3648	192	0	3456
<i>aflow40b.mps</i>	1442	2728	1364	0	1364
<i>atm20-100.mps</i>	4380	6480	2220	0	4260
<i>bab1.mps</i>	60680	61152	61152	0	0
<i>bab3.mps</i>	23069	393800	393800	0	0
<i>bab5.mps</i>	4964	21600	21600	0	0
<i>bley_xl1.mps</i>	175620	5831	5831	0	0
<i>bnatt350.mps</i>	4923	3150	3150	0	0
<i>bnatt400.mps</i>	5614	3600	3600	0	0
<i>cdma.mps</i>	9095	7891	4235	0	3656
<i>circ10-3.mps</i>	42620	2700	2700	0	0
<i>core4872-1529.mps</i>	4875	24656	24645	0	11
<i>datt256.mps</i>	11077	262144	262144	0	0
<i>ds-big.mps</i>	1042	174997	174997	0	0
<i>enlight14.mps</i>	196	392	196	196	0
<i>enlight15.mps</i>	225	450	225	225	0
<i>enlight16.mps</i>	256	512	256	256	0
<i>ex10.mps</i>	69608	17680	17680	0	0
<i>ex9.mps</i>	40962	10404	10404	0	0
<i>f2000.mps</i>	10500	4000	4000	0	0
<i>germanrr.mps</i>	10779	10813	5288	5286	239
<i>hanoi5.mps</i>	16399	3862	3862	0	0
<i>ivu52.mps</i>	2116	157591	157591	0	0
<i>lectsched-1-obj.mps</i>	50108	28718	28236	482	0
<i>lectsched-1.mps</i>	50108	28718	28236	482	0

<i>lectsched-2.mps</i>	30738	17656	17287	369	0
<i>lectsched-3.mps</i>	45262	25776	25319	457	0
<i>lrta120.mps</i>	14521	3839	120	119	3600
<i>mining.mps</i>	661133	348921	348920	0	1
<i>momentum1.mps</i>	42680	5174	2349	0	2825
<i>momentum2.mps</i>	24237	3732	1808	1	1923
<i>momentum3.mps</i>	56822	13532	6598	1	6933
<i>n15-3.mps</i>	29494	153140	0	780	152360
<i>nb10tb.mps</i>	150495	73340	14124	2756	56460
<i>neos-1140050.mps</i>	3795	40320	38640	0	1680
<i>neos-1224597.mps</i>	3276	3395	3150	245	0
<i>neos-1429212.mps</i>	58726	416040	54756	0	361284
<i>neos-631710.mps</i>	169576	167056	167056	0	0
<i>neos-686190.mps</i>	3664	3660	3600	60	0
<i>neos-738098.mps</i>	25849	9093	8946	0	147
<i>neos-785912.mps</i>	1714	1380	1380	0	0
<i>neos-807456.mps</i>	840	1635	1635	0	0
<i>neos-820146.mps</i>	830	600	600	0	0
<i>neos-820157.mps</i>	1015	1200	1200	0	0
<i>neos-826650.mps</i>	2414	5912	5792	0	120
<i>neos-849702.mps</i>	1041	1737	1737	0	0
<i>neos-859770.mps</i>	2065	2504	2504	0	0
<i>neos-885524.mps</i>	65	91670	91670	0	0
<i>neos-932816.mps</i>	30823	21007	20566	0	441
<i>neos-935627.mps</i>	7859	10301	7522	0	2779
<i>neos-937511.mps</i>	8158	11332	8562	0	2770
<i>neos-</i>	9251	11646	8876	0	2770

<i>937815.mps</i>					
<i>neos-952987.mps</i>	354	31329	31329	0	0
<i>neos-984165.mps</i>	6962	8883	6478	0	2405
<i>neos808444.mps</i>	18329	19846	19846	0	0
<i>neos858960.mps</i>	132	160	160	0	0
<i>net12.mps</i>	14021	14115	1603	0	12512
<i>netdiversion.mps</i>	119589	129180	129180	0	0
<i>ns1116954.mps</i>	131991	12648	7482	0	5166
<i>ns1208400.mps</i>	4289	2883	2880	0	3
<i>ns1456591.mps</i>	1997	8399	8000	19	380
<i>ns1631475.mps</i>	24496	22696	22470	211	15
<i>ns1644855.mps</i>	40698	30200	10000	0	20200
<i>ns1685374.mps</i>	44121	10000	10000	0	0
<i>ns1696083.mps</i>	11063	7982	7982	0	0
<i>ns1745726.mps</i>	4687	3208	3208	0	0
<i>ns1758913.mps</i>	624166	17956	17822	0	134
<i>ns1769397.mps</i>	5527	3772	3772	0	0
<i>ns1778858.mps</i>	10666	4720	4720	0	0
<i>ns1853823.mps</i>	224526	213440	213440	0	0
<i>ns1854840.mps</i>	143616	135754	135280	474	0
<i>ns1904248.mps</i>	149437	38458	38416	0	42
<i>ns1905797.mps</i>	51884	18192	17676	4	512
<i>ns1905800.mps</i>	8289	3228	3030	3	195
<i>ns1952667.mps</i>	41	13264	0	13264	0
<i>ns2118727.mps</i>	163354	167440	159514	0	7926
<i>ns2122603.mps</i>	24754	19300	7588	0	11712
<i>ns2137859.mps</i>	206726	103361	103041	0	320
<i>ns894236.mps</i>	8218	9666	9666	0	0
<i>ns894786.mps</i>	16794	27278	27278	0	0
<i>ns894788.mps</i>	2279	3463	3463	0	0
<i>ns903616.mps</i>	18052	21582	21582	0	0
<i>nsr8k.mps</i>	6284	38356	32040	0	6316
<i>opm2-z10-s2.mps</i>	160633	6250	6250	0	0
<i>opm2-z11-s8.mps</i>	223082	8019	8019	0	0
<i>opm2-z12-s14.mps</i>	319508	10800	10800	0	0

<i>opm2-z12-s7.mps</i>	319508	10800	10800	0	0
<i>p2m2p1m1p0n100.mps</i>	1	100	100	0	0
<i>probportfolio.mps</i>	302	320	300	0	20
<i>rail01.mps</i>	46843	117527	117527	0	0
<i>rail02.mps</i>	95791	270869	270869	0	0
<i>rail03.mps</i>	253905	758775	758775	0	0
<i>rmine10.mps</i>	65274	8439	8439	0	0
<i>rmine14.mps</i>	268535	32205	32205	0	0
<i>rocll-7-11.mps</i>	37215	16101	15851	0	250
<i>rocll-9-11.mps</i>	47533	20679	20361	0	318
<i>rococoC12-111000.mps</i>	10776	8619	8432	187	0
<i>satellites2-60-fs.mps</i>	16516	35378	34324	0	1054
<i>satellites2-60.mps</i>	20916	35378	34324	0	1054
<i>satellites3-40-fs.mps</i>	35553	81681	79961	0	1720
<i>satellites3-40.mps</i>	44804	81681	79961	0	1720
<i>sct32.mps</i>	5440	9767	6396	1332	2039
<i>shipsched.mps</i>	45554	13594	10549	0	3045
<i>shs1023.mps</i>	133944	444625	1296	440899	2430
<i>sing161.mps</i>	455631	770102	733244	0	36858
<i>sing359.mps</i>	437116	713762	674643	0	39119
<i>stockholm.mps</i>	57346	20644	962	0	19682
<i>stp3d.mps</i>	159488	204880	204880	0	0
<i>triptim2.mps</i>	14427	27326	20771	6548	7
<i>triptim3.mps</i>	14939	28440	21621	6812	7
<i>uc-case3.mps</i>	52003	37749	11256	0	26493
AVERAGE	66848.99	68453.91	56794.44	4337.55	7321.92

A.6 MIPLIB 2010 Subset of Equality-Containing MIP Models

Using Perpendicular General Disjunctions

The table below lists the subset of 43 equality-containing models in which the default general algorithm (provided in Table 3) triggered the use of at least one perpendicular general disjunction. Models in which only perpendicular general disjunctions were triggered by all algorithms in Experiments 3 and 4.B (Sections 7.3 and 7.4.2) are indicated in the columns.

MIP Model	Experiment 3 algorithms trigger only perpendicular disjunctions?	Experiment 4.B algorithms trigger only perpendicular disjunctions?
neos-1396125.mps	no	no
neos-506428.mps	no	no
neos-799711.mps	yes	yes
neos-826841.mps	yes	no
neos-777800.mps	yes	yes
dano3mip.mps	yes	yes
neos-935769.mps	no	yes
acc-tight5.mps	no	no
gmu-35-40.mps	no	no
neos-941262.mps	no	no
bienst2.mps	yes	yes
mcsched.mps	yes	yes
neos-506422.mps	yes	yes
danoimt.mps	yes	yes
neos-911880.mps	yes	yes
co-100.mps	yes	yes
eilA101.2.mps	yes	yes
neos-1109824.mps	no	no
neos-555424.mps	yes	no
nag.mps	yes	yes
csched010.mps	yes	yes
30n20b8.mps	no	no
neos-476283.mps	no	no
csched008.mps	yes	yes
neos-1620770.mps	yes	no

neos-941313.mps	yes	yes
neos-520729.mps	yes	yes
glass4.mps	yes	yes
markshare_5_0.mps	no	yes
gmu-35-50.mps	no	no
mkc.mps	no	no
blp-ar98.mps	no	yes
blp-ic97.mps	yes	yes
leo1.mps	no	no
dc1c.mps	no	no
eilB101.mps	yes	yes
biella1.mps	no	no
leo2.mps	yes	yes
eil33.2.mps	yes	yes
csched007.mps	yes	yes
harp2.mps	yes	yes
neos-1440225.mps	yes	yes
air04.mps	yes	yes