Interior Point Cutting Plane Methods in Integer Programming

by

Joe Naoum-Sawaya

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Management Sciences

Waterloo, Ontario, Canada, 2011

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis presents novel approaches that use interior point concepts in solving mixed integer programs. Particularly, we use the analytic center cutting plane method to improve three of the main components of the branch-and-bound algorithm: cutting planes, heuristics, and branching.

First, we present an interior point branch-and-cut algorithm for structured integer programs based on Benders decomposition. We explore using Benders decomposition in a branch-and-cut framework where the Benders cuts are generated using the analytic center cutting plane method. The algorithm is tested on two classes of problems: the capacitated facility location problem and the multicommodity capacitated fixed charge network design problem. For the capacitated facility location problem, the proposed approach was on average 2.5 times faster than Benders-branch-and-cut and 11 times faster than classical Benders decomposition. For the multicommodity capacitated fixed charge network design problem, the proposed approach was 4 times faster than Benders-branch-and-cut while classical Benders decomposition failed to solve the majority of the tested instances.

Second, we present a heuristic algorithm for mixed integer programs based on interior points. As integer solutions are typically in the interior, we use the analytic center cutting plane method to search for integer feasible points within the interior of the feasible set. The algorithm searches along two line segments that connect the weighted analytic center and two extreme points of the linear programming relaxation. Candidate points are rounded and tested for feasibility. Cuts aimed to improve the objective function and restore feasibility are then added to displace the weighted analytic center until a feasible integer solution is found. The algorithm is composed of three phases. In the first, points along the two line segments are rounded gradually to find integer feasible solutions. Then in an attempt to improve the quality of the solutions, the cut related to the bound constraint is updated and a new weighted analytic center is found. Upon failing to find a feasible integer solution, a

second phase is started where cuts related to the violated feasibility constraints are added. As a last resort, the algorithm solves a minimum distance problem in a third phase. For all the tested instances, the algorithm finds good quality feasible solutions in the first two phases and the third phase is never called.

Finally, we present a new approach to generate good general branching constraints based on the shape of the polyhedron. Our approach is based on approximating the polyhedron using an inscribed ellipsoid. We use Dikin's ellipsoid which we calculate using the analytic center. We propose to use the disjunction that has a minimum width on the ellipsoid. We use the fact that the width of the ellipsoid in a given direction has a closed form solution in order to formulate a quadratic problem whose optimal solution is a thin direction of the ellipsoid. While solving a quadratic problem at each node of the branch-and-bound tree is impractical, we use a local search heuristic for its solution. Computational testing conducted on hard integer problems from MIPLIB and CORAL showed that the proposed approach outperforms classical branching.

Acknowledgements

I am most grateful to my supervisor Prof. Samir Elhedhli for his continuous guidance and support throughout my Ph.D. studies. He has been an inspiration to my career. His advice and encouragement during my master's and Ph.D. have been invaluable to me.

I would like to thank Prof. David Fuller, Prof. Fatma Gzara, Prof. Henry Wolkowicz, and Prof. John Mitchell for serving on my Ph.D. committee and for their insightful comments that improved this research.

I would like to acknowledge the financial support from the Natural Sciences and Engineering Research Council through the Alexander Graham Bell Canada Graduate Scholarship and the Ontario Graduate Scholarship Program for funding my Ph.D. research.

I wish to express my gratitude to Ali, Sachin, Navneet, Emre, Simon, Mark, Tiffany, Elspeth, Christie, Hessa, Eman, Juan, Jim, Alex, Johannes, and the dozens of friends and colleagues at the Department of Management Sciences. Thank you for creating an enjoyable environment.

I also would like to thank Prof. Michael Jünger who provided me with the opportunity to spend two terms at the University of Cologne. It has been an exceptional experience to work with him and his research group. Thank you to Christoph Buchheim, Frauke Liers, Andreas Schmutzer, Gregor Pardella, Martin Gronemann, Thomas Lange, Sven Mallach, Daniel Plümpe, Göntje Teuchert, Frank Baumann, Merijam Percan, Michael Schulz, and Andrea Wagner for making my stay in Cologne fruitful and memorable.

I am greatly indebted to Bissan Ghaddar who has provided me with unconditional love and support throughout the best and the most difficult times at Waterloo. Without her, I wouldn't have achieved this success.

Finally, I am forever indebted to my parents. Thank you Fouad, Liliane, and my brother Hadi for your constant care, support, and encouragements.

Table of Contents

List of Tables

List of Figures

Chapter 1

Large Scale Linear and Integer Programming

Nowadays, linear and integer programming are widely used techniques with many real life applications. Leonid Kantorovich was among the first to use linear programming to solve a resource allocation problem where a factory's production is maximized subject to resource availability [44]. Kantorovich modeled this problem as a linear function that is maximized subject to a set linear inequalities, a technique later called linear programming. Around the same period, Tjalling Koopmans independently used linear programming to model problems arising in economics. The breakthrough in linear programming is attributed to George Dantzig after the invention of the Simplex algorithm in the late 1940's. The simplex algorithm made it possible to solve large linear problems inspired from real applications. Although the simplex algorithm works well on average, in the worst case it needs an exponential number of iterations to find the optimal solution. In 1979, Leonid Khachiyan used the ellipsoid method that was developed by David Yudin and Arkadi Nemirovski [77] and independently by Naum Shor [68]. The ellipsoid method was proven to be a polynomial time algorithm for linear programming. In contrast to the simplex method,

the performance of the ellipsoid method was always close to the worst case which although polynomial, is typically large. Khachiyan's algorithm answered an important question on the complexity of solving linear programs, however the simplex method outperformed the ellipsoid method in practice. The second breakthrough after the simplex came in 1984 with the invention of Karmarkar's interior point algorithm which sparked the rise of the field of interior point methods. Interior point methods often outperform the simplex algorithm though mainly on large problems.

Mixed Integer programming models are linear programs where some of the decision variables are restricted to be integers. The importance of integer programming stems from the fact that many applications are modeled using integer rather than continuous variables. Although many general purpose software are available to solve integer programs, large scale integer programming continues to be a challenge. The main solution approaches for integer programming are based on branch-and-bound [47]. The main idea of branch-and-bound is to successively divide an integer program into relaxations that are easy to solve until one of the relaxations provides the optimal solution of the original problem. Initially at the root node, the branch-and-bound starts by solving a relaxation of the original problem. Bounds on the optimal solution are found using the solution of the relaxation and by attempting to identify a feasible solution using heuristics. If the lower and upper bounds match, then an optimal solution has been found, otherwise, a branching mechanism is applied to tighten the relaxations by splitting the feasible region of the original problem into typically two regions each forming a new problem that is solved recursively. For a branch-and-bound to be effective, the branching mechanism should create problems that are easy to solve while the bounding mechanism should yield tight bounds that help in fathoming the nodes early in the branching tree. A common approach for integer programming is to create problems based on linear programming relaxations; mainly due to the ability of solving large linear programs efficiently.

In order to improve the bound provided by each relaxation, cutting planes are applied to strengthen the linear programming relaxation. Crowder et al. [20] were the first to explore the combination of cutting planes and branch-and-bound in a branch-and-cut algorithm. The idea behind branch-and-cut is to apply branch-and-bound where the subproblems are tightened by adding linear inequalities that separate the fractional solutions without cutting off any of the feasible integer solutions. The most famous of these cutting planes are Gomory cutting planes [38]. Nowadays, all commercial solvers use Branch-and-cut as the standard approach to solve mixed integer programming problems.

In addition to using linear programming relaxations in branch-and-bound, Lagrangian relaxation has also been widely used. At each node of the branching tree, the dual of the Lagrangian master problem is solved using a column approach where a restriction of the problem is first solved and columns are added as needed. The combination of column generation with branch-and-bound is known as branch-and-price. Branch-and-price was first introduced in [24] and has been a major topic of research in recent years. Mainly the research focused on improving the column generation [25] and on designing new branching rules [64, 73, 72, 9].

Many large scale integer programs possess structure that can be exploited to speed up the solutions process. Benders decomposition, Dantzig-Wolfe decomposition, and Lagrangian relaxation are the most commonly used methods to exploit structured programs. These methods are based on solving a series of smaller subproblems such that the solution converges to that of the original problem. Benders decomposition and Lagrangian relaxation are cutting plane methods where the series of problems that are solved are relaxations of the original model and cutting planes are added to improve the relaxation. On the other hand, Dantzig-Wolfe is a column generation method where a series of restrictions of the original model are solved and columns are added to improve the restriction.

Besides these methods which are devised to find optimal solutions, there has been an increasing interest in the development of heuristics that can find good feasible solutions.

Heuristics are typically applied to find solutions for hard problems often with no guarantees on the quality of the solution. Greedy heuristics find feasible solutions using information about the problem structure. For example for the Traveling Salesman Problem, a simple greedy heuristic can start from any city and then selects the next closest city to visit next. Metaheuristics are based on intelligent search techniques which try to explore the search space efficiently while avoiding getting stuck in local optimal solutions. Among the metaheuristics that have been presented in literature, the most widely used are genetic algorithms [41], simulated annealing [57], and tabu search [30, 31]. Another class of general heuristics is based on using solutions of the relaxation of a mixed integer program, namely the optimal solution of the LP relaxation, to find feasible solutions. Although simple rounding is in most cases unsuccessful, algorithms such as the feasibility pump [12, 2] which iteratively rounds candidate points of the LP relaxation was successful on general mixed integer programs and it has been implemented in a number of commercial solvers.

1.1 Thesis Content and Contribution

In this research, we study the application of interior point methods in integer programming. Some of the early attempts to use interior-points for integer programming are due to Mitchell and Todd [60] and Mitchell [58] who use a primal-dual predictor-corrector interior point method in a cutting plane method to solve integer programs. Being aware that warm starting is key to a successful method, the interior point method is terminated early at a central interior point that is later used to warm start the solution methodology after cuts are added [13, 36, 59]. They obtained encouraging results on the maximum cut problem and the linear ordering problem. Elhedhli and Goffin [25] use an interior-point cutting plane method within a branch-and-bound framework that uses a Lagrangian bound. The bound is found using the Analytic Center Cutting Plane method (ACCPM). The resulting interior point branch-and-price warm starts the solution of child nodes using a dual interior-

point method as would classical branch-and-bound do using the dual simplex. Gzara and Goffin [39] propose an interior-point branch-and-bound-and-cut that exploits warm starting through primal and dual interior-point methods. They use a cut and column generation scheme based on ACCPM to solve the centralized network design problem on directed graphs.

In this thesis, we explore new venues in the application of interior point methods in integer programming by focusing on the three main components of a branch-and-cut algorithm: cutting planes, heuristics, and branching. Particularly, we propose to use the analytic center cutting plane method in a Benders-Branch-and-cut framework, in a heuristic to find feasible solutions for general mixed integer programs, and finally to derive general branching disjunctions. In particular, we exploit warm starting within the Benders framework by using Benders decomposition within branch-and-bound instead of solving the master problem as an integer program. This is achieved through proving that the Benders cuts are global and hence apply to any node in the branch-and-bound tree. While this makes it possible to reuse Benders cuts, we also reduce their number by using ACCPM and proving that such cuts are pareto optimal. Motivated by the observation that it is more likely that rounding an interior point to the nearest integer will result in a feasible integer solution compared to an extreme point, we present a new iterative heuristic based on rounding the analytic center of a series of relaxations of the mixed integer program. Cuts related to the violated constraints are added to improve the relaxation and to increase the chances of finding a feasible integer solution. Finally to improve the quality of branching, we devise generalized branching disjunctions based on the use of analytic centers. At each node of the branch-and-bound tree, the polyhedron is first approximated using Dikin's ellipsoid which is centered at the analytic centre. The branching disjunction corresponding to the minimum-width of the ellipsoid is then used for branching.

The rest of the document is organized as follows. In Chapter 2, we review two cutting plane methods: Kelley's cutting plane method and the analytic center cutting plane method. In

Chapter 3, we discuss Dantzig-Wolfe decomposition and Benders decomposition for solving mixed integer problems. In Chapter 4, we present the integration of the analytic center cutting plane method in Dantzig-Wolfe decomposition. In Chapter 5, we present a branch-and-cut algorithm based on Benders decomposition and the analytic center cutting plane method. In Chapter 6, we present a new heuristic for finding feasible solutions for mixed integer programs based on interior points. In Chapter 7, we present the new approach to generate general branching disjunctions. Finally, Chapter 8 concludes and highlights future research.

Chapter 2

Cutting Plane Methods

Cutting plane methods are used to solve general convex optimization problems. They are based on the assumption that given a candidate solution, a separation oracle either asserts that the candidate point is a feasible point and returns a supporting hyperplane of the epigraph of the objective function or returns a separating hyperplane. The choice of the candidate point is important for the performance of the cutting plane algorithm. In this chapter, we revisit two cutting plane methods: Kelley's cutting plane method [46] and the analytic center cutting plane method (ACCPM) [35].

2.1 Kelley's Cutting Plane Method

Kelley's cutting plane method was introduced in [46] as a method for solving convex non-differentiable problems. It is based on the fact that every convex function $f(y)$ can be approximated by a set of piecewise linear functions that are tangent to $f(y)$ at a subset of finite points $y_i,\ i \in I$. To show this, we use the fact that a function $f(y)$ is convex over a

set Y if

$$f(y_1) \geq f(y_2) + h^T(y_2)(y_1 - y_2) \ \forall y_1, y_2 \in Y$$

where $h(y)$ is the subgradient of $f(y)$. If $f(y)$ is differentiable at point y_2 then $h(y_2) = \nabla f(y_2)$ where $\nabla f(y)$ is the gradient of $f(y)$. The function $f(y)$ can be approximated as

$$f(y) \backsimeq \max_{i \in I} \{f(y_i) + h^T(y_i)(y - y_i)\}. \tag{2.1}$$

Using (2.1), every convex problem

$$\begin{aligned} &\min \ f(y) \\ &\text{s.t. } \ g(y) \leq 0 \end{aligned} \tag{2.2}$$

can be approximated as

$$\begin{aligned} &\min\{\max_{i \in I}\{f(y_i) + h_1^T(y_i)(y - y_i)\}\} \\ &\text{s.t. } \ \max_{j \in J}\{g(y_j) + h_2^T(y_j)(y - y_j)\} \leq 0 \end{aligned}$$

where $h_1(y)$ and $h_2(y)$ are the gradients of $f(y)$ and $g(y)$ respectively, which is equivalent to

$$\min \theta \tag{2.3}$$

$$\text{s.t. } \ f(y_i) + h_1^T(y_i)(y - y_i) \leq \theta, \ \forall i \in I \tag{2.4}$$

$$g(y_j) + h_2^T(y_j)(y - y_j) \leq 0 \ \forall j \in J. \tag{2.5}$$

Problem (2.3)-(2.5) is an approximation problem of (2.2). This approximation gets sharper as more constraints of type (2.4) and (2.5) are added, however problem (2.3)-(2.5) is a linear

problem that can be easier to solve than problem (2.2). Kelley's cutting plane starts by an initial relaxation of problem (2.3)-(2.5) where a subset of constraints (2.4)-(2.5) are considered. The optimal solution $(\overline{y}, \overline{\theta})$ of the relaxation is then used to generate a cut of the form (2.5) if $g(\overline{y}) > 0$ otherwise a cut of the form (2.4) is generated. By relaxation $(\overline{y}, \overline{\theta})$ provides a lower bound on the optimal solution of problem (2.2). If $g(\overline{y}) \leq 0$ then $(\overline{y})$ is a feasible solution of problem (2.2) and thus it provides an upper bound. The gap between the upper and lower bounds is typically used as a stopping criterion for Kelley's cutting plane algorithm. A summary of Kelley's cutting plane algorithm follows:

1. Initialization: Get an initial upper UB and lower bound LB on the optimal solution $f(y^*)$.
2. Start with an initial relaxation of problem (2.3)-(2.5)
3. While $UB - LB > \epsilon$
   - 3.1 Find the optimal solution $(\overline{y}, \overline{\theta})$ of the relaxation.
     - 3.1.1 Update lower bound $LB = \overline{\theta}$
   - 3.2 if $g(\overline{y}) > 0$
     - 3.2.1 generate a cut of the form (2.5)
   - 3.3 Else
     - 3.3.1 generate a cut of the form (2.4)
     - 3.3.2 Update upper bound $UB = \min\{UB, f(\overline{y})\}$
4. Go back to Step 3.

A key difference between the various cutting plane methods is the choice of the query point which is used to generate the cuts. In what follows we present three variants: the largest inscribed sphere method, the volumetric center method, and the analytic center cutting plane method.

2.2 Largest Inscribed Sphere Method

In the largest inscribed sphere cutting plane method, the query point is the center of the largest inscribed sphere of the localization set. The center of the largest inscribed sphere in the bounded polyhedron $\{x : a_i^T x \leq b_i,\ \forall i \in I\}$ can be found by solving the following linear problem

$$\begin{aligned} &\max \sigma \\ &\text{s.t. } a_i^T x + \|a_i\|\sigma \leq b_i \ \forall i \in I, \end{aligned}$$

where $\|.\|$ denotes the Euclidean norm. Details of the largest inscribed sphere method can be found in [26].

2.3 Volumetric Center Method

In contrast to using the center of the largest inscribed sphere, the query point in the volumetric center cutting plane method is the center of the largest inscribed ellipsoid. The center of the largest inscribed ellipsoid in the bounded polyhedron $\{x : a_i^T x \leq b_i,\ \forall i \in I\}$ can be found by solving

$$\min \log(\det(\sum_i^I \frac{a_i a_i^T}{(b_i - a_i^T x)^2})).$$

Details of the volumetric center cutting plane method can be found in [70].

2.4 Analytic Center Cutting Plane Method

Similar to other cutting plane methods, the analytic center cutting plane method (ACCPM) is closely related to Kelley's cutting plane method. The main difference is the choice of the point from which cutting planes are generated. In Kelley's cutting plane method, cutting planes are generated from the optimal solution of the relaxation, while in ACCPM, the cutting planes are generated from a point that is located in the center of the bounded feasible region of the relaxation. Namely, given the general piecewise linear approximation (2.3)-(2.5), the analytic center of the bounded polyhedron

$$\mathcal{F} = \left\{ \begin{array}{c} \theta \leq \theta_u \\ f(y_i) + h_1^T(y_i)(y - y_i) \leq \theta, \ \forall i \in I \\ g(y_j) + h_2^T(y_j)(y - y_j) \leq 0 \ \forall j \in J \\ y_l \leq y \leq y_u \end{array} \right\} \tag{2.6}$$

is used to generate the cutting planes. The upper bound θ_u on the objective function value and the box constraints $y_l \leq y \leq y_u$ ensure that the polyhedron $\mathcal{F}$ is bounded. The analytic center (y^{ac}, θ^{ac}) of $\mathcal{F}$ is used to generate a cut of the form (2.5) if $g(y^{ac}) > 0$ otherwise a cut of the form (2.4) is generated. By relaxation, the dual analytic center (x^{ac}) corresponding to (y^{ac}, θ^{ac}) provides a lower bound on the optimal solution of problem (2.2), while if $g(y^{ac}) \leq 0$ then (y^{ac}) is a feasible solution of problem (2.2) and thus it provides an upper bound. The gap between the upper and lower bounds is used as a stopping criterion for ACCPM.

2.4.1 Calculating the Analytic Center

The main step in ACCPM is the calculation of the analytic center. For ease of exposition we consider finding the analytic center associated with the following general linear problem:

$$\begin{aligned} &\min\ b^T y \\ &\text{s.t. } A_q^T y \leq c_q, \end{aligned}$$

where $A_q^T y \leq c_q$ contain the box constraints $y_l \leq y \leq y_u$. The analytic center corresponding to the polyhedron

$$\mathcal{F} = \left\{ \begin{array}{l} b^T y \leq \theta_u \\ A_q^T y \leq c_q \end{array} \right\} \tag{2.7}$$

is the point that maximizes the product of the distances from the boundaries of $\mathcal{F}$. Usually a weight equal to the number of constraints is associated with the upper bound constraint $b^T y \leq \theta_u$ to push the analytic center away from the upper bound. In the general case, we can associate a positive weight vector v where v_i is the weight of constraint i. It is worth noting that repeating a constraint is equivalent to incrementing its corresponding weight [34]. The analytic center of $\mathcal{F}$ is then the optimal solution of the primal-dual pair

$$\begin{aligned} (D):\ \varphi_D(s) = \max\ &\sum_{i=0}^{m} v_i \ln s_i \\ \text{s.t. } &A^T y + s = c \\ &s > 0 \end{aligned}$$

$$(P): \ \varphi_P(x) = \max \ -c^T x + \sum_{i=0}^{m} v_i \ln x_i$$
$$\text{s.t. } Ax = 0,$$
$$x > 0.$$

where $A^T = \begin{bmatrix} b^T \\ A_q^T \end{bmatrix}$ and $c = \begin{bmatrix} \theta_u \\ c_q \end{bmatrix}$. The necessary and sufficient first order optimality conditions are

$$Sx = v, \tag{2.8}$$
$$Ax = 0, \qquad x > 0, \tag{2.9}$$
$$A^T y + s = c, \qquad s > 0, \tag{2.10}$$

where S is the diagonal matrix with diagonal s. Three possible methods can be used to calculate the analytic center: A dual Newton algorithm, a primal Newton algorithm, and a primal-dual Newton algorithm. Next we present the details of each of the Newton methods.

Dual Newton Algorithm

The dual Newton method starts from a solution $(\overline{y}, \overline{s})$ that is strictly feasible to $A^T y + s = c$ and moves along a direction (d_y, d_s) that improves the dual potential function

$$\varphi_D(s) = \max \ \sum_{i=0}^{m} v_i \ln s_i$$

and maintains feasibility. The first and second derivatives of $\varphi_D(s)$ are $VS^{-1}e$ and $-VS^{-2}$ where V is the diagonal matrix of v. The Newton directions are the optimal solution of

$$\begin{aligned} &\max\ (VS^{-1}e)d_s - \frac{1}{2}d_s^T VS^{-2}d_s, \\ &\text{s.t. } A^T d_y + d_s = 0, \\ &\quad \overline{s} + d_s > 0. \end{aligned}$$

Let x be the dual variable associated with $A^T d_y + d_s = 0$ then the first order optimality conditions are

$$VS^{-1}e - VS^{-2}d_s - x = 0, \tag{2.11}$$

$$A^T d_y + d_s = 0, \tag{2.12}$$

$$Ax = 0, \tag{2.13}$$

which solve for

$$d_y = -(AVS^{-2}A^T)^{-1}AVS^{-1}e \tag{2.14}$$

$$d_s = A^T(AVS^{-2}A^T)^{-1}AVS^{-1}e \tag{2.15}$$

$$x = VS^{-1}e - VS^{-2}A^T(AVS^{-2}A^T)^{-1}AVS^{-1}e \tag{2.16}$$

The norm of the gradient of $\varphi_D(s)$, $g(s) = VS^{-1}d_s$, can be used as a proximity measure of how far the current solution is from the analytic center. The dual Newton algorithm follows

Initialization Start with a strictly feasible solution (y, s) such that $A^T y + s = c$ and $s > 0$ and a stopping criterion ϵ. Initialize $g(s) = VS^{-1}A^T(AVS^{-2}A^T)^{-1}AVS^{-1}e$.

While $\|g(s)\| \geq \epsilon$

1. $d_y = -(AVS^{-2}A^T)^{-1}AVS^{-1}e$
2. $d_s = A^T(AVS^{-2}A^T)^{-1}AVS^{-1}e$
3. $\alpha = \text{argmax}\{\varphi_D(s + \alpha d_s) : s + \alpha d_s \geq 0\}$
4. $s = s + \alpha d_s$
5. $y = y + \alpha d_y$
6. $g(s) = VS^{-1}A^T(AVS^{-2}A^T)^{-1}AVS^{-1}e$

End

Primal Solution $x = VS^{-1}e - VS^{-2}A^T(AVS^{-2}A^T)^{-1}AVS^{-1}e$

Next we present the primal Newton algorithm.

Primal Newton Algorithm

The primal Newton method starts from a solution $\overline{x}$ that is strictly feasible to $Ax = 0$ and moves in a direction d_x that improves the objective function

$$\varphi_P(s) = \max \ -c^T x + \sum_{i=0}^{m} v_i \ln x_i$$

and maintains feasibility. The first and second derivatives of $\varphi_P(s)$ are $NX^{-1}e - c$ and $-NX^{-2}$ and the Newton direction is the optimal solution of

$$\max \ (NX^{-1}e - c)d_x - \frac{1}{2}d_x^T(NX^{-2})d_x \tag{2.17}$$

$$\text{s.t. } Ad_x = 0 \tag{2.18}$$

$$\overline{x} + d_x > 0. \tag{2.19}$$

Let y be the dual variable associated with $Ad_x = 0$ then the first order optimality conditions are

$$NX^{-1}e - c - NX^{-2}d_x + A^T y = 0, \tag{2.20}$$

$$Ad_x = 0. \tag{2.21}$$

From (2.20) and (2.21) we get

$$(AN^{-1}X^2A^T)y - AN^{-1}X^2c + AXe = 0.$$

Since $AXe = 0$ then

$$y = (AN^{-1}X^2A^T)^{-1}AN^{-1}X^2c.$$

Finally, from (2.20) we get

$$\begin{aligned} d_x &= (NX^{-2})^{-1}(A^T y - c + NX^{-1}e) \\ &= (NX^{-2})^{-1}(A^T(AN^{-1}X^2A^T)^{-1}AN^{-1}X^2c - c + NX^{-1}e) \end{aligned} \tag{2.22}$$

The norm of the gradient of $\varphi_P(x)$, $g(x) = (NX^{-1}e - c)d_x$, can be used as a proximity measure. The primal Newton algorithm follows

Initialization Start with a strictly feasible solution x such that $Ax = 0$ and $x > 0$ and a stopping criterion ϵ. Initialize $g(x) = (NX^{-1}e-c)(NX^{-2})^{-1}(A^T(AN^{-1}X^2A^T)^{-1}AN^{-1}X^2c-c + NX^{-1}e)$.

While $\|g(x)\| \geq \epsilon$

1. $d_x = (NX^{-2})^{-1}(A^T(AN^{-1}X^2A^T)^{-1}AN^{-1}X^2c - c + NX^{-1}e)$
2. $\alpha = \text{argmax}\{\varphi_P(x + \alpha d_x) : x + \alpha d_x \geq 0\}$

3. $x = x + \alpha d_x$
4. $g(x) = (NX^{-1}e - c)(NX^{-2})^{-1}(A^T(AN^{-1}X^2A^T)^{-1}AN^{-1}X^2c - c + NX^{-1}e)$

End

Dual Solution

1. $y = (AN^{-1}X^2A^T)^{-1}AN^{-1}X^2c$
2. $s = c - A^Ty$

Next we present the primal-dual Newton algorithm.

Primal-Dual Newton Algorithm

The primal-dual Newton algorithm starts from a solution $(\overline{x}, \overline{y}, \overline{s})$ that is strictly feasible to $Ax = 0$ and $A^Ty + s = c$ and moves in a direction (d_x, d_y, d_s) that improves the potential function $\varphi_D(s) + \varphi_P(x)$ and maintains feasibility. The optimal solution is the point that satisfies

$$\begin{aligned} &Sx = v, && \\ &Ax = 0, && x > 0, \\ &A^Ty + s = c, && s > 0. \end{aligned}$$

The Newton direction (d_x, d_y, d_s) is

$$\begin{bmatrix} S & 0 & X \\ A & 0 & 0 \\ 0 & -A^T & -I \end{bmatrix} \begin{bmatrix} d_x \\ d_y \\ d_s \end{bmatrix} = \begin{bmatrix} v - Sx \\ 0 \\ 0 \end{bmatrix}$$

which solve for

$$
\begin{aligned}
d_y &= (AS^{-1}XA^T)^{-1}(AS^{-1})(Sx - v), \\
d_s &= -A^T d_y, \\
d_x &= S^{-1}(v - Sx - Xd_s).
\end{aligned}
$$

The norm of $g(x, s) = v - Sx$ can be used as a proximity measure. The primal-dual Newton method follows

Initialization Start from a strictly feasible solution (x, y, s) such that $Ax = 0,\ A^T y + s = c$ and a stopping criterion ϵ.

While $\|v - Sx\| \geq \epsilon$

1. $d_y = (AS^{-1}XA^T)^{-1}(AS^{-1})(Sx - v)$
2. $d_s = -A^T d_y$
3. $d_x = S^{-1}(v - Sx - Xd_s)$
4. $\alpha = \text{argmax}\{\varphi_P(x + \alpha d_x) + \varphi_D(s + \alpha d_s) : x + \alpha d_x \geq 0,\ s + \alpha d_s \geq 0\}$
5. $x = x + \alpha d_x$
6. $y = y + \alpha d_y$
7. $s = s + \alpha d_s$

End

If a primal feasible solution is available then a primal Newton algorithm is used. If a dual feasible solution is available the a dual Newton method is used. If both a primal and a dual feasible solution are available, then a primal-dual Newton algorithm is used.

Chapter 3

Decomposition Methods

Many large-scale optimization problems have a structure formed by the non-zero coefficients in the constraints matrix. In particular, two types of block angular structures are identified: block-angular structure with linking constraints (Figure 3.1) and block angular structure with linking variables also known as dual block angular structure (Figure 3.2). By relaxing the constraints in a block angular structure with linking constraints, the solution is obtained by solving small independent problems that are easier than the original problem. Similarly by fixing the variables in a block angular structure with linking variables, the solution is obtained by solving small independent problems that are easier than the original problem. Two decomposition algorithms are used to tackle structured problem: Dantzig-Wolfe decomposition [22] and Benders decomposition [11]. We note that problems with a block angular structure with linking constraints and linking variables are solved using cross decomposition [71] which is a combination of Dantzig-Wolfe and Benders decomposition.

In this Chapter, we review Dantzig-Wolfe decomposition and Benders decomposition algorithms in the context of solving structured integer programs. We discuss the theoretical and the implementation details.

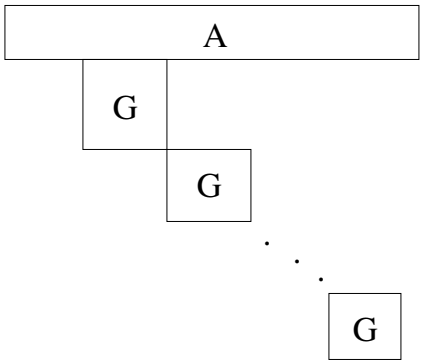

Figure 3.1: Block Angular Structure with Linking Constraints

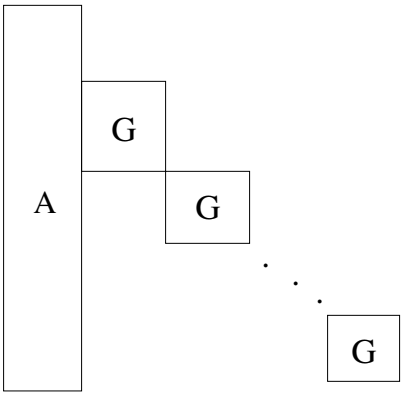

Figure 3.2: Block Angular Structure with Linking Variables

3.1 Dantzig-Wolfe Decomposition

Although Dantzig-Wolfe decomposition was presented to solve problems with a block angular structure with linking constraints, in principal it can be applied to any integer program. Let us first introduce the general linear programming model

$$\min\ b^T x, \tag{3.1}$$

$$\text{s.t. } Ax \geq c, \tag{3.2}$$

$$Gx \geq h, \tag{3.3}$$

$$x \geq 0, \tag{3.4}$$

$$x_t \text{ is integer } \forall t \in T. \tag{3.5}$$

Every feasible solution of the set $\mathcal{S} = \{Gx \geq h,\ x \geq 0,\ x_t \text{ is integer } \forall t \in T\}$ can be expressed as a convex combination of all the extreme points and a non-negative combination of all the extreme rays of the set $\mathcal{S}$ [61]. Given P and R the index sets of the extreme points x^i and the extreme rays μ^j of $\mathcal{S}$ respectively; $x \in \mathcal{S}$ can be expressed as

$$x = \sum_{i \in P} u_i x^i + \sum_{j \in R} v_j \mu^j, \tag{3.6}$$

$$\sum_{i \in P} u_i = 1, \tag{3.7}$$

$$u_i \in \{0, 1\}\ \forall i \in P, \tag{3.8}$$

$$v_j \geq 0\ \forall j \in R. \tag{3.9}$$

Replacing (3.6)-(3.9) in (3.1)-(3.5) we get

$$\min \sum_{i \in P} u_i (b^T x^i) + \sum_{j \in R} v_j (b^T \mu^j), \tag{3.10}$$

$$\text{s.t.} \sum_{i \in P} u_i (A x^i) + \sum_{j \in R} v_j (A \mu^j) \geq c, \tag{3.11}$$

$$\sum_{i \in P} u_i = 1, \tag{3.12}$$

$$u_i \in \{0, 1\} \ \forall i \in P, \tag{3.13}$$

$$v_j \geq 0 \ \forall j \in R. \tag{3.14}$$

Problem (3.10)-(3.14) is then solved by branch-and-bound. In general, problem (3.10)-(3.14) contains a very large number of extreme points and extreme rays therefore the LP relaxation corresponding to each node of the branch-and-bound tree is solved by column generation. At a particular node, the following full master problem is solved

$$\begin{aligned}
\text{[DW-FMP]: min} & \sum_{i \in P_b} u_i (b^T x^i) + \sum_{j \in R_b} v_j (b^T \mu^j), \\
\text{s.t.} & \sum_{i \in P_b} u_i (A x^i) + \sum_{j \in R_b} v_j (A \mu^j) \geq c, \\
& \sum_{i \in P_b} u_i = 1, \\
& u_i \geq 0 \ \forall i \in P_b, \\
& v_j \geq 0 \ \forall j \in R_b,
\end{aligned}$$

where P_b and R_b denote the sets of extreme points x^i and extreme rays μ^i such that $x^i \in P$, $\mu^i \in R$ and x^i and μ^i satisfy all the branching constraints. In what follows, we describe the column generation algorithm applied to [DW-FMP]. Starting with a subset of extreme

points and rays $\overline{P} \subseteq P_b$ and $\overline{R} \subseteq R_b$, the restricted Dantzig-Wolfe master problem is

$$\text{[DW-MP]: min} \sum_{i \in \overline{P}} u_i (b^T x^i) + \sum_{j \in \overline{R}} v_j (b^T \mu^j), \tag{3.15}$$

$$\text{s.t.} \sum_{i \in \overline{P}} u_i (A x^i) + \sum_{j \in \overline{R}} v_j (A \mu^j) \geq c, \tag{3.16}$$

$$\sum_{i \in \overline{P}} u_i = 1, \tag{3.17}$$

$$u_i \geq 0 \ \forall i \in \overline{P}, \tag{3.18}$$

$$v_j \geq 0 \ \forall j \in \overline{R}. \tag{3.19}$$

The optimal solution of [DW-MP] is an optimal solution of [DW-FMP] if all the variables of [DW-FMP] have non-negative reduced costs. Given an optimal solution for [DW-MP] with $\overline{\pi}$ and $\overline{\alpha}$ the corresponding dual variables associated with constraints (3.16) and (3.17) respectively, to check if the optimal solution of [DW-MP] is optimal for [DW-FMP] corresponds to checking if all the reduced costs $b^T x^i - \overline{\pi} A x^i - \overline{\alpha}, \ \forall i \in P_b$ are non-negative. This corresponds to solving

$$\text{[DW-SP]: min } b^T x - \overline{\pi} A x, \tag{3.20}$$

$$\text{s.t. } Gx \geq h, \tag{3.21}$$

$$x \geq 0, \tag{3.22}$$

$$\text{branching constraints.} \tag{3.23}$$

If [DW-SP] is unbounded, then an extreme ray is identified and added to [DW-MP]. Otherwise, given x^* the optimal solution of [DW-SP], if $b^T x^* - \overline{\pi} A x^* - \overline{\alpha} \geq 0$ then x^* is the optimal solution of [DW-FMP]. Otherwise x^* is an extreme point that is added to [DW-MP]. Since [DW-MP] is a restriction of [DW-FMP], then the optimal solution of [DW-MP]

that is obtained at every iteration of the Dantzig-Wolfe decomposition algorithm is an upper bound on the optimal solution of [DW-FMP].

If the optimal solution at a particular node leads to a fractional $\overline{x}_t = \sum_{i \in P} u_i x_t^i + \sum_{j \in R} v_j \mu_t^j$ then branching is done by creating two children nodes by adding one of the branching constraints

$$x_t \leq \lfloor \overline{x}_t \rfloor \text{ and } x_t \geq \lfloor \overline{x}_t \rfloor + 1. \tag{3.24}$$

Other branching rules such as the Ryan-and-Foster rule [64] have also been used (see Elhedhli and Goffin [25]).

Finally, we note that columns are not only generated at the root node of the branch-and-bound tree but at every node of the tree leading to a branch-and-price algorithm. Furthermore, cutting planes such as Gomory cuts can also be combined with branch-and-price which lead to branch-and-price-and-cut algorithms.

3.2 Benders Decomposition

Benders decomposition is typically applied to problems with a block-angular structure with linking variables of the form

$$\begin{aligned} \min\ & b^T x + d^T y \\ \text{s.t. } & A_1^T y \leq c_1, \\ & A_2^T y + G^T x \geq c_2, \\ & y \geq 0 \text{ and integer}, \\ & x \geq 0. \end{aligned} \tag{3.25}$$

By projecting problem (3.25) on the space defined by the linking variables y, the resulting problem is:

$$\begin{aligned} &\min_{y} \{d^T y + \min_{x}\{b^T x | G^T x \geq c_2 - A_2^T y, x \geq 0\}\} \\ &\text{s.t. } A_1^T y \leq c_1, \\ &\quad y \geq 0 \text{ and integer.} \end{aligned} \tag{3.26}$$

The inner minimization problem is replaced by the dual maximization problem as follows:

$$\begin{aligned} &\max (c_2 - A_2^T y)\lambda \\ &\text{s.t. } G\lambda \leq b, \\ &\quad \lambda \geq 0. \end{aligned}$$

Let H^P and H^R be the sets of extreme points and extreme rays of the set $\{\lambda | G\lambda \leq b, \lambda \geq 0\}$. The problem formulated in (3.26) is equivalent to

$$\min d^T y + \theta \tag{3.27}$$

$$\text{s.t. } A_1^T y \leq c_1, \tag{3.28}$$

$$(c_2 - A_2^T y)\lambda \leq \theta \quad \lambda \in H^P, \tag{3.29}$$

$$(c_2 - A_2^T y)\mu \leq 0 \quad \mu \in H^R, \tag{3.30}$$

$$y \geq 0 \text{ and integer.} \tag{3.31}$$

Starting with an initial empty set of extreme rays and extreme points, the cutting plane algorithm solves the relaxed IP master problem

$$\begin{aligned} &\min\ d^T y \\ &\text{s.t. } A_1^T y \leq c_1, \\ &\qquad y \geq 0 \text{ and integer.} \end{aligned} \tag{3.32}$$

Its solution $\overline{y}$ is used to generated optimality and feasibility cuts by solving the primal subproblem:

$$\begin{aligned} &\min\ b^T x \\ &\text{s.t. } G^T x \geq c_2 - A_2^T \overline{y}, \\ &\qquad x \geq 0, \end{aligned}$$

or equivalently, its dual

$$\begin{aligned} &\max\ (c_2 - A_2^T \overline{y})\lambda \\ &\text{s.t. } G\lambda \leq b, \\ &\qquad \lambda \geq 0. \end{aligned} \tag{3.33}$$

If the primal subproblem is infeasible or equivalently the dual subproblem is unbounded, then a feasibility cut is generated from the dual extreme ray of unboundedness, by solving

the auxiliary subproblem

$$\begin{aligned} \max\ & 0 \\ \text{s.t. } & (c_2 - A_2^T \overline{y})\lambda = 1, \\ & G\lambda \leq 0, \\ & \lambda \geq 0. \end{aligned} \tag{3.34}$$

Finally, the generated optimality or feasibility cuts are appended to the master problem (7.20)-(7.24) and the algorithm reiterates until an optimal solution is found. As the relaxed master problem is a relaxation of the original problem, it provides a lower bound on the optimal solution of the original problem. Furthermore, given the optimal solution $(\overline{y}, \overline{\theta})$ of the master problem and solving the subproblem for $\overline{x}$, $(\overline{x}, \overline{y})$ is feasible to the original problem, and hence the objective function evaluated at the point $(\overline{x}, \overline{y})$ gives an upper bound. The master and the subproblem are solved iteratively until the gap between the upper and lower bounds is sufficiently small.

Similar to Dantzig-Wolfe decomposition where the columns are generated based on an extreme point which is the optimal solution of the master problem, Kelley's cutting plane method is also used in Benders decomposition to generate the Benders cuts. In the following chapter, we discuss using interior points to generate the columns in Dantzig-Wolfe using the analytic center cutting plane method. Then in Chapter 5, we apply the analytic center cutting plane method to Benders decomposition in a branch-and-cut setting.

Chapter 4

ACCPM in Branch-and-Price

Kelley's cutting plane algorithm has been shown to have a poor performance in a number of cases. For instance, when the subproblem has alternative optimal solutions, column generation achieves a better performance when an interior point is used compared to when an extreme point is used. The reason appears to be that with an interior point method, the query point which is contained in the optimal face provides a better representation compared to an extreme point method where the query point is a vertex of the optimal face [42]. Another observation is that Kelley's cutting plane method lacks stability where the solutions move significantly after cutting planes are added [14]. Additionally, Kelley's cutting plane suffers from the tailing-off effect where typically excessive computational time is spent on closing the duality gap in the final iterations. The analytic center cutting plane method was shown to achieve good performance on a wide range of problems [35, 5, 37]. In this chapter, we discuss the application of the analytic center cutting plane method in Branch-and-Price first discussed in [25].

4.1 The Analytic Center Cutting Plane Method in Branch-and-Price

In classical branch-and-price methods, columns are generated using a dual extreme point of the restricted master problem. Elhedhli and Goffin [25] were the first to use ACCPM in a branch-and-price approach where columns are generated based on a central point. ACCPM is used to solve the column generation problem at each node of branch-and-price tree. At each iteration, the analytic center of the [DW-MP] is calculated and is used to generate a column from [DW-SP]. Similar to classical branch-and-bound, the node is fathomed if the optimal solution of the corresponding problem is feasible to the original problem, the node is infeasible, or the lower bound is worse than the incumbent. A Ryan-Foster branching rule [64] is used where the branching constraints are added to the subproblem to prevent the generation of infeasible columns, an approach that tends to be efficient.

4.2 Calculating the Analytic Center After Adding Cuts

As detailed in Section 2.4.1, the analytic center is calculated using a Newton method whose performance is typically affected by the starting point. After adding cuts, a primal feasible solution is easily recovered and a primal Newton algorithm is used to calculate a new analytic center. To recover a primal feasible solution Elhedhli and Goffin [25] use the fact that adding a cut to the dual problem corresponds to adding a column to the primal problem and primal feasibility is recovered by taking a step d_x from the current solution $\overline{x}$ to get $x = \overline{x} + \alpha d_x$ into the interior of the feasible region. The old analytic center $(\overline{y}, \overline{s}, \overline{x})$

satisfies the optimality conditions

$$\overline{X}\overline{s} = \overline{v} \tag{4.1}$$

$$A\overline{x} = 0, \qquad \overline{x} > 0 \tag{4.2}$$

$$A^T\overline{y} + \overline{s} = \overline{c}, \qquad \overline{s} > 0. \tag{4.3}$$

The new optimality conditions after adding cuts $B^T y \leq r$ are

$$Xs = v \tag{4.4}$$

$$\Delta\ h = e \tag{4.5}$$

$$Ax + B\delta = 0, \qquad x > 0 \tag{4.6}$$

$$A^T y + s = c, \qquad s > 0. \tag{4.7}$$

$$B^T y + h = r, \qquad h > 0. \tag{4.8}$$

where δ is the dual variable corresponding to $B^T y \leq r$. Using $x = \overline{x} + d_x$, $y = \overline{y} + d_y$, and $s = \overline{s} + d_s$, we get

$$d_y = -(A\overline{X}\ \overline{S}^{-1}A^T)^{-1}(B\delta + A\overline{S}^{-1}(v - \overline{v} + \overline{X}(c - \overline{c}))) \tag{4.9}$$

$$d_s = -A^T d_y + (c - \overline{c}) \tag{4.10}$$

$$d_x = -\overline{S}^{-1}\overline{X}d_s + \overline{S}^{-1}(v - \overline{v}) \tag{4.11}$$

$$h = \Delta^{-1}e \tag{4.12}$$

where Δ is the diagonal matrix with diagonal δ, which is the optimal solution corresponding to maximizing

$$\max\ \sum_j \ln \delta_j - \frac{1}{2}\delta^T B^T (A\overline{X}\ \overline{S}^{-1}A^T)^{-1}B\delta + g^T\delta. \tag{4.13}$$

where $g = B^T\overline{y} - r - B^T(A\overline{X}\ \overline{S}^{-1}A^T)AS^{-1}(v - \overline{v} + \overline{X}(c - \overline{c}))$. Problem (4.13) is solved using a Newton method and the resulting point is primal feasible (see Elhedhli and Goffin [25] for the complete details).

4.3 Calculating the Analytic Center After Branching

After branching, the columns of the parent node are partitioned into two sets such that only the columns that satisfy the branching constraint are included in each of the children nodes. Therefore a new analytic center should be calculated after dropping columns/cuts. The analytic center of the parent node remains feasible and a dual Newton algorithm is used to calculate the new analytic center as in Section 2.4.1.

4.4 Conclusion

In this chapter, we revisited the ACCPM branch-and-price which was first presented in [25]. ACCPM is used to solve the column generation problem at each node of the branch-and-price tree. Methods to recover feasibility to warm-start the Newton method have been discussed. In the following chapter, we present the integration of ACCPM in a Benders based branch-and-cut algorithm.

Chapter 5

ACCPM in Branch-and-Cut

[1] Benders decomposition transforms a mixed integer problem with preferably a dual block angular structure into two problems: a mixed integer master problem involving the integer variables with an additional continuous variable, and a linear subproblem involving the continuous variables. The master problem and the subproblem are then solved iteratively in a cutting plane algorithm. At each iteration, the optimal solution of a relaxation of the master problem is used to solve the subproblem. If the subproblem is feasible, a Benders optimality cut is added to the master problem, otherwise a Benders feasibility cut is added. The iterative algorithm is stopped when a solution that doesn't violate any optimality or feasibility cut is identified.

The Benders decomposition algorithm suffers from a major computational bottleneck. The master problem, which is solved repeatedly, is an integer problem that gets more difficult as more cuts are added. In addition, it is not clear how to fully exploit warm starting when resolving an integer program after a cut is added. What is typically done is to add a bound constraint on the objective function based on the previous optimal solution.

[1]The material in this chapter has been published in J. Naoum-Sawaya and S. Elhedhli, "An Interior-Point Benders based Branch-and-Cut Algorithm for Mixed Integer Programs", Annals of Operations Research, DOI: 10.1007/s10479-010-0806-y.

A number of attempts have been made to alleviate this problem. Geoffrion and Graves [29] discuss the effect of the problem formulation on improving the computational efficiency due to the tighter linear programming (LP) relaxation within the branch-and-bound algorithm. Mervet [56] illustrates the effect of adding an initial set of valid cuts to the master problem to tighten the feasible region of the master problem. In the same context, McDaniel and Devine [55] propose the use of cuts generated from the LP relaxation of the master problem to construct an initial set of valid cuts. For integer problems, Cote and Laughton [18] suggest that instead of solving the master problem to optimality, heuristics may be used to find integer feasible solutions in order to generate feasibility cuts. The concept of pareto optimal cuts, introduced by Magnanti and Wong [52] , looks for the tightest cut to add to the master problem when the subproblem has multiple optimal solutions. It turned out to be efficient in solving network type of problems. Rei et al. [63] describe a local branching approach to be used within Benders decomposition to improve the lower and upper bounds at each iteration. After observing that in some cases, the cuts generated by the Benders subproblem are low-density cuts and their contribution to strengthening the Benders master problem is limited, Saharidis et al. [66] propose a novel way to generate multiple cuts by solving an auxiliary problem based on the solution of the master problem. The proposed algorithm is tested on the scheduling of crude oil and the scheduling of multi-product, multi-purpose batch plants and was shown to significantly decrease the number of iterations and the computational time. Recently, Saharidis and Ierapetritou [65] presented a maximum feasible subsystem cut generation approach where at each iteration of the Benders algorithm, an extra cut is generated to restrict the value of the objective function of the master problem. This strategy focuses on the particular case where more feasibility than optimality Benders cuts are produced. Testing on scheduling problems for multi-purpose multi-product batch plants revealed that the proposed methodology significantly reduces the computational time.

In this work, we try to exploit warm starting within the Benders framework by using Benders decomposition within branch-and-bound instead of solving master problems as integer programs. This is achieved through proving that the Benders cuts are global and hence apply to any node in the branch-and-bound tree. While this makes it possible to reuse Benders cuts, we also reduce their number by using ACCPM and proving that such cuts are pareto optimal. With this we achieve very good speed ups that are beyond reach using classical Benders decomposition.

5.1 Benders Decomposition Branch-and-Cut algorithm

To overcome the disadvantage of repeatedly solving the integer master problem, we propose to integrate Benders decomposition in a branch-and-cut algorithm. The Benders cut are dynamically generated within the branch-and-bound when the master problem is solved.

Branch-and-cut is a combination of branch-and-bound and cutting planes and is one of the main approaches to solve mixed integer programming. General cutting planes such as Gomory cuts, disjunctive cuts, lift-and-project cuts, split cuts, and mixed-integer-rounding cuts are all used in branch-and-cut frameworks in the state of the art software. We explore the use of Benders decomposition in a branch-and-cut framework.

In branch-and-cut, the algorithm proceeds by solving a linear problem at each node of the branch-and-bound tree resulting in a solution that might violate the integer requirements of the original mixed integer problem. Valid cuts are added to eliminate the fractional solutions. After adding cuts, if the solution remains fractional, branching is performed. In the context of Benders decomposition, we propose to use branch-and-cut to solve the Benders master problem [MP]. The Benders optimality and feasibility cuts, (3.29) and (3.30) respectively, are not known beforehand and are therefore added as they are identified along the branch-and-cut tree. At the root node, starting with an empty set of feasibility and optimality cuts, the linear programming (LP) relaxation of the Benders master problem

[MP] is solved. Its optimal solution is used to solve the Benders subproblem and generate cuts. If some of the generated cuts are violated by the optimal solution of the LP relaxation, the violated cuts are added to the linear master problem which is resolved. Otherwise if no valid cut is identified, the algorithm proceeds by branching on a variable that doesn't satisfy the integrality requirements. The iterative algorithm is then resumed by solving the resulting linear Benders master problem to find a potential solution and then solving the Benders subproblem to identify valid cuts. The main advantage of the Benders-branch-and-cut over the classical iterative Benders algorithm (Section 3.2) is the elimination of the necessity to solve multiple integer programs from scratch. Rather than repeatedly applying branch-and-bound to solve the integer master problem to optimality, adding a Benders cut and restarting the branch-and-bound from scratch; Benders framework is warm started by integrating the Benders cut generation within branch-and-bound, therefore exploring a single branch-and-bound tree.

Two types of cuts are generated in a branch-and-cut algorithm: local and global cuts. Local cuts are only valid for the problem solved at a particular node and all its descendants, whereas global cuts are valid for the original problem and therefore are valid at every node of the tree. Generally, global cuts have a greater advantage compared to local cuts as they are generated once and are used at all nodes. The following propositions show that the Benders optimality and feasibility cuts are global cuts and are valid at every node of the tree.

Proposition 1. *Consider the following master problem that is solved at a particular node of the branch-and-cut tree:*

$$[LMP] : \min\ d^T y + \theta \tag{5.1}$$

$$s.t.\ A_1^T y \leq c_1, \tag{5.2}$$

$$(c_2 - A_2^T y)\lambda \leq \theta \quad \lambda \in H^P, \tag{5.3}$$

$$(c_2 - A_2^T y)\mu \leq 0 \quad \mu \in H^R, \tag{5.4}$$

$$y \geq 0, \tag{5.5}$$

$$\textit{Branching Constraints,} \tag{5.6}$$

and $(\overline{y}, \overline{\theta})$ *is a feasible solution whose corresponding dual subproblem is bounded and has an optimal solution* $\overline{\lambda}$*. Then the corresponding Benders optimality cut*

$$(c_2 - A_2^T y)\overline{\lambda} \leq \theta$$

is valid to the original IP master problem [MP] and is therefore a global cut.

Proof: Consider the optimal solution (y^*, θ^*) of the original IP master problem [MP]. The Benders optimality cut corresponding to y^* is

$$(c_2 - A_2^T y)\lambda^* \leq \theta \tag{5.7}$$

where $(c_2 - A_2^T y^*)\lambda^*$ is the optimal objective function value of the dual subproblem when $y = y^*$. Consider $(\overline{y}, \overline{\theta})$ which is a feasible solution to problem [LMP]. The Benders optimality cut corresponding to $\overline{y}$ is

$$(c_2 - A_2^T y)\overline{\lambda} \leq \theta \tag{5.8}$$

where $\overline{\lambda}$ is the optimal solution and $(c_2 - A_2^T\overline{y})\overline{\lambda}$ is the optimal objective function of the dual subproblem when $y = \overline{y}$. Since both $\overline{\lambda}$ and λ^* are feasible solutions of the subproblem, and since λ^* is an optimal solution when $y = y^*$, then

$$(c_2 - A_2^T y^*)\overline{\lambda} \leq (c_2 - A_2^T y^*)\lambda^*. \tag{5.9}$$

Since (y^*, θ^*) is the optimal solution of the original IP master problem [MP] then (y^*, θ^*) is feasible to (5.7) and therefore

$$(c_2 - A_2^T y^*)\lambda^* \leq \theta^*. \tag{5.10}$$

From (5.9) and (5.10) we get

$$(c_2 - A_2^T y^*)\overline{\lambda} \leq \theta^*,$$

proving that (y^*, θ^*) is feasible to (5.8). □

Proposition 2. *Consider the master problem [LMP] that is solved at a particular node of the branch-and-cut tree with optimal solution $(\overline{y}, \overline{\theta})$ for which the dual subproblem is unbounded and has an extreme ray $\overline{\mu}$, the corresponding Benders feasibility cut*

$$(c_2 - A_2^T y)\overline{\mu} \leq 0 \tag{5.11}$$

is valid to the original IP master problem [MP] and is therefore a global cut.

Proof: $\overline{\mu}$ is the extreme ray of the dual subproblem and therefore any solution that satisfies $(c_2 - A_2^T y)\overline{\mu} > 0$ makes the dual subproblem unbounded and is not feasible to the full master problem. In particular the optimal solution (y^*, θ^*) of the original IP master problem [MP] should satisfy $(c_2 - A_2^T y^*)\overline{\mu} \leq 0$ and therefore (y^*, θ^*) is feasible to $(c_2 - A_2^T y)\overline{\mu} \leq 0$. □

Propositions 1 and 2 show that the Benders cuts are global cuts. We exploit this property by constructing a cut pool to which we add all the Benders cuts that are generated at

every node of the branch-and-cut tree. After branching, the cut pool is used to warm start the child nodes. A summary of the Benders-branch-and-cut algorithm follows:

1. Initialization: Set the incumbent $Z_U = +\infty$. Set the cut pool $P = \emptyset$.
   Initialize the root node with the LP relaxation of problem (3.32).
2. Select a non fathomed node from the tree. If none exist, stop. Else,
   - 2.1 Initialize $z_l = -\infty$, $z_u = +\infty$. z_l, and z_u are the lower and upper bounds on the LP relaxation at the current node of the tree.
   - 2.2 While $|z_u - z_l| > \varepsilon$, $z_l < Z_U$, and the problem is feasible
     - 2.2.1 Find the optimal solution $(\overline{y}, \overline{\theta})$ of the master problem (5.1)-(5.6) at the selected node.
     - 2.2.2 Solve the subproblem (3.33) with $y = \overline{y}$ and generate a Benders cut.
     - 2.2.3 Add the generated Benders cut to the master problem at the selected node and to the cut pool P.
     - 2.2.4 If $(\overline{y}, \overline{\theta})$ doesn't violate the generated cut, update the upper bound $z_u = \min\{z_u, d^T\overline{y} + \overline{\theta}\}$. As detailed in Section 3.2, we can also update $z_u = \min\{z_u, b^T\overline{x} + d^T\overline{y}\}$, where $\overline{x}$ is the dual solution of the subproblem, (if subproblem (3.33) is bounded and feasible).
     - 2.2.5 Update the lower bound $z_l = \max\{z_l, d^T\overline{y} + \overline{\theta}\}$.
   - 2.3 End While.
   - 2.4 If $z_l > Z_U$ or if the master problem (5.1)-(5.6) at the selected node is infeasible, fathom the node.
     - 2.4.1 Go back to Step 2.
   - 2.5 If $(\overline{y}, \overline{\theta})$ satisfies the integrality constraints, fathom the node and update $Z_U = \min\{Z_U, d^T\overline{y} + \overline{\theta}\}$. Otherwise, create a new node by branching as in classical branch-and-bound and append the cuts that are in the cut pool P to the problem that is created in the new node.
3. Go back to Step 2.

As the query point returned by the master problem is an extreme point, we refer to this branch-and-cut algorithm as Kelley-Benders-branch-and-cut. A critical factor in the efficiency of Kelley-Benders-branch-and-cut is the quality of the generated Benders cuts. Step 2.2 is stopped when $(\overline{y}, \overline{\theta})$ doesn't violate any of the Benders cuts, which is equivalent to $z_u = z_l$. Often the Benders cuts, as generated in the classical Benders decomposition, are shallow cuts because they are generated using the extreme points of the master problem. Furthermore a large number of cuts is usually required in the early stages of the branch-and-bound tree before a solution $(\overline{y}, \overline{\theta})$ that doesn't violate any Benders cut is reached. Alternatively, Step 2.2 may be stopped before all the required cuts are added. This leads however to additional unnecessary branching that would be fathomed if all the required cuts were added in preceding nodes.

In the following section, we propose to use the analytic center cutting plane method (ACCPM) instead of Kelley's cutting plane method in Step 2.2. The motivation behind using ACCPM is that it generates tighter Benders cuts and improves the performance of the branch-and-cut algorithm.

5.2 The Analytic Center Cutting Plane Method in Benders Decomposition

In Step 2.2 of Kelley-Benders-branch-and-cut, we substitute Kelley's cutting plane by ACCPM. The main advantage of using ACCPM is that the analytic center often resides near the geometric center of a polyhedron and most likely, the generated Benders cuts separate a big portion of the polyhedron. Therefore, the optimal solution is found in fewer iterations with fewer Benders cuts. Equivalently when ACCPM is used, a smaller number of problems are solved at each branch of the branch-and-bound tree compared to those solved when Kelley's cutting plane is used. Step 2.2 is then modified as follows. First

the analytic center (y^{ac}, θ^{ac}) of problem (5.1)-(5.6) which is solved at each node of the branch-and-cut tree is found. Solving subproblem (3.33) with $y = y^{ac}$, a Benders cut is generated. If (y^{ac}, θ^{ac}) does not violate the generated Benders cut, then the upper bound z_u is updated as $z_u = \min\{z_u, d^T y^{ac} + \theta^{ac}\}$. Furthermore by relaxation, any dual feasible point to problem (5.1)-(5.6) gives a lower bound z_l. To see this, let us consider δ^{ac} to be the dual solution corresponding to (y^{ac}, θ^{ac}). For ease of exposition, let us assume that the objective function of the dual of problem (5.1)-(5.6) is denoted by $f^T\delta$. Let δ^* be the optimal dual solution and (y^*, θ^*) be the optimal primal solution then

$$z_l = f^T\delta^{ac} \leq f^T\delta^* = d^T y^* + \theta^*$$

The upper bound z_u and the lower bound z_l can then be used as a stopping criterion for Step 2.2 which is summarized as follows

2.2 While $|z_u - z_l| > \varepsilon$, $z_l < Z_U$, and the problem is feasible

- 2.2.1 Find the analytic center (y^{ac}, θ^{ac}) of the master problem (5.1)-(5.6) at the selected node.
- 2.2.2 Solve the subproblem (3.33) with $y = y^{ac}$ and generate a Benders cut.
- 2.2.3 Add the generated Benders cut to the master problem at the selected node and to the cut pool P.
- 2.2.4 If (y^{ac}, θ^{ac}) doesn't violate the generated cut, update the upper bound $z_u = \min\{z_u, d^T y^{ac} + \theta^{ac}\}$.
- 2.2.5 Update the lower bound $z_l = \max\{z_l, f^T\delta^{ac}\}$.

2.3 End While.

We will refer to this branch-and-cut algorithm as ACCPM-Benders-branch-and-cut.

5.3 Calculating the Analytic Center after Adding Cuts

Although an approach similar to [25] can be used to restart a primal Newton method to calculate a new analytic center after adding cuts, we introduce a new warm starting strategy based on the assumption that box constraints are always present in the constraints set. For the problem $\min\{b^T y : A_q^T y \leq c_q, y^l \leq y \leq y^u\}$ with the primal-dual analytic center $(\overline{y}, \overline{x}, \overline{x}^+, \overline{x}^-)$, the primal analytic center satisfies the primal feasibility condition

$$[A,\ I,\ -I] \begin{bmatrix} \overline{x} \\ \overline{x}^+ \\ \overline{x}^- \end{bmatrix} = 0,$$

where I is an identity matrix and $A = [b, A_q]$. Adding a cut $B^T y \leq r$ to the problem corresponds to adding a variable δ to the primal space. The new primal feasibility condition is

$$[A,\ B,\ I,\ -I] \begin{bmatrix} x \\ \delta \\ x^+ \\ x^- \end{bmatrix} = 0.$$

A primal feasible solution can be obtained from the old primal analytic center by taking a step $(d_x,\ d_\delta,\ d_{x^+},\ d_{x^-})$ from the old analytic center such that

$$A(\overline{x} + d_x) + B(d_\delta) + I(\overline{x}^+ + d_{x^+}) - I(\overline{x}^- + d_{x^-}) = 0$$

which reduces to

$$A(d_x) + B(d_\delta) + I(d_{x^+}) - I(d_{x^-}) = 0.$$

A feasible step $(d_x,\ d_\delta,\ d_{x^+},\ d_{x^-})$ can then be found as follows:

- Choose $d_\delta > 0$.
- Set $d_x = 0$.
- If $(B)_i > 0$ then set $(d_{x^-})_i = (B)_i(d_\delta)_i$, where $()_i$ denotes the i^{th} element of the vector.
- else set $(d_{x^+})_i = -(B)_i(d_\delta)_i$.

The resulting primal feasible solution is used to start the primal newton method.

5.4 Analytic Centers and Pareto-Optimality

In many instances of Benders decomposition, the subproblem is usually degenerate and several cuts can be generated and added to the master problem [52]. Selecting and adding the best cuts significantly improves the performance of the algorithm. First, we revisit the following concept introduced by Magnanti and Wong [52]. A cut

$$(c_2 - A_2^T y)\overline{\lambda} \leq \theta$$

is dominated by

$$(c_2 - A_2^T y)\lambda^* \leq \theta$$

if

$$(c_2 - A_2^T y)\overline{\lambda} \leq (c_2 - A_2^T y)\lambda^*, \ \forall y \in Y$$

there exists a $y \in Y$ such that

$$(c_2 - A_2^T y)\overline{\lambda} < (c_2 - A_2^T y)\lambda^*.$$

A cut is pareto-optimal if it is not dominated by any other cut. Pareto-optimal cuts are obtained by solving an auxiliary subproblem. The following theorem proves that a Benders cut that is generated from the analytic center is pareto-optimal.

Theorem 1. *A Benders cut* $(c_2 - A_2^T y)\lambda^* \leq \theta$ *that is generated from the analytic center of the master problem is pareto-optimal.*

Proof: The proof is similar to the proof of Theorem 1 in [52]. Let y^{ac} be the analytic center corresponding to the master problem. Let λ^* be an optimal solution of the corresponding subproblem. Suppose that the generated Benders cut

$$(c_2 - A_2^T y)\lambda^* \leq \theta \tag{5.12}$$

is not pareto-optimal. Then there exists a cut

$$(c_2 - A_2^T y)\overline{\lambda} \leq \theta \tag{5.13}$$

that dominates (5.12), where $\overline{\lambda}$ is an optimal solution of the subproblem that corresponds to y^{ac}. Since (5.13) dominates (5.12) then

$$(c_2 - A_2^T y)\overline{\lambda} \geq (c_2 - A_2^T y)\lambda^*, \ \forall y \in Y; \tag{5.14}$$

and there exists a $\overline{y} \in Y$ where

$$(c_2 - A_2^T \overline{y})\overline{\lambda} > (c_2 - A_2^T \overline{y})\lambda^*. \tag{5.15}$$

Furthermore, since both λ^* and $\overline{\lambda}$ are optimal solutions of the subproblem corresponding to y^{ac}, then

$$(c_2 - A_2^T y^{ac})\overline{\lambda} = (c_2 - A_2^T y^{ac})\lambda^*. \tag{5.16}$$

Since y^{ac} is the analytic center of the master problem and hence is in the relative interior of Y then there exists a scalar $\alpha > 1$ where $w = (1-\alpha)\overline{y} + \alpha y^{ac}$ and $w \in Y$. Multiplying (5.15) by $(1-\alpha)$ and (5.16) by α and adding, we get

$$(1-\alpha)(c_2 - A_2^T\overline{y})\overline{\lambda} + \alpha(c_2 - A_2^T y^{ac})\overline{\lambda} < (1-\alpha)(c_2 - A_2^T\overline{y})\lambda^* + \alpha(c_2 - A_2^T y^{ac})\lambda^*,$$

which reduces to

$$(c_2 - A_2^T w)\overline{\lambda} < (c_2 - A_2^T w)\lambda^*. \tag{5.17}$$

(5.17) contradicts with (5.14) showing that the assumption that (5.12) is not pareto-optimal is not valid. □

Theorem 1 shows that the Benders cuts that are generated by ACCPM are pareto-optimal cuts, eliminating the need to solve an auxiliary problem as in [52].

5.5 Implementation and Testing

In this section, we test the proposed ACCPM-Benders-branch-and-cut on two classical applications of Benders decomposition: the capacitated facility location problem (CFLP) [75] and the multicommodity capacitated fixed charge network design problem (MCFND) [17].

The branch-and-cut algorithm and the analytic center cutting plane algorithms were implemented in C. The branch-and-cut algorithm uses a depth first search strategy. As detailed in the algorithm description in Section 5.2, the incumbent is used as a stopping criterion for the ACCPM algorithm which is stopped as soon as the lower bound exceeds the incumbent value. Additionally, a stopping criterion $\epsilon = 10^{-4}$ is used for the ACCPM algorithm. CPLEX 11.0 was used to solve the LP subproblems. In the classical Benders decomposition

the master problem, which is an IP problem, was solved using a branch-and-bound algorithm that uses a depth first search strategy similar to the branch-and-cut search strategy. The branch-and-bound and the branch-and-cut have identical implementations except that in the branch-and-bound the Benders cuts are not generated. Additionally no preprocessing was used and with the exception of the Benders cuts in the branch-and-cut, no other cutting planes are used in the solvers.

To assess the efficiency of the proposed methodologies, the ACCPM-Benders-branch-and-cut algorithm (ABBC) is compared to the classical Kelley-Benders-branch-and-cut (KBBC) algorithm, the classical Benders decomposition of Section 3.2, and the classical Benders decomposition with the addition of pareto-optimal cuts. The computational results are conducted on a SUNBLADE 2500 workstation with a 1.6 GHz processor and 2 Gb of memory.

5.5.1 The Capacitated Facility Location Problem

The capacitated facility location problem (CFLP) is an important problem that has a wide range of applications including telecommunication network planning and production planning. The use of CFLP for our testbed is motivated by the fact that [75] presents a simple algorithm for generating pareto-optimal cuts thus allowing us to compare between the presented branch-and-cut algorithm and the classical Benders decomposition with and without pareto-optimal cuts. A formulation for the capacitated facility location problem

as given in [75] is:

$$
\begin{aligned}
\min & \sum_{j=1}^{n} f_j y_j + \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \\
\text{s.t.} & \sum_{j=1}^{n} x_{ij} = d_i \qquad \forall i = 1, \ldots, m \\
& x_{ij} \leq d_i y_j \qquad \forall i = 1, \ldots, m, j = 1, \ldots, n, \\
& \sum_{i=1}^{m} x_{ij} \leq s_j y_j \qquad j = 1, \ldots, n, \\
& x_{ij} \geq 0,\ y_j \in \{0, 1\}\ \forall i = 1, \ldots, m, j = 1, \ldots, n.
\end{aligned}
$$

The set of potential facility locations and the set of customers are indexed by i and j respectively. Each facility has a capacity s_j and a fixed cost f_j. Each customer i has a demand d_i and a variable cost c_{ij} if serviced from facility j. The binary variable y_j takes the value of 1 when facility location j is selected, and x_{ij} indicated the amount of products allocated to customer i from facility j. Fixing the facility location variables $y = \overline{y}$ leads to the following primal Benders subproblem

$$
\begin{aligned}
\sum_{j=1}^{n} f_j \overline{y}_j + \min & \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \\
\text{s.t.} & \sum_{j=1}^{n} x_{ij} = d_i \qquad \forall i = 1, \ldots, m \\
& x_{ij} \leq d_i \overline{y}_j \qquad \forall i = 1, \ldots, m, j = 1, \ldots, n, \\
& \sum_{i=1}^{m} x_{ij} \leq s_j \overline{y}_j \qquad j = 1, \ldots, n, \\
& x_{ij} \geq 0,\ \forall i = 1, \ldots, m, j = 1, \ldots, n.
\end{aligned}
$$

The corresponding dual subproblem is

$$\sum_{j=1}^{n} f_i \overline{y}_j + \max \sum_{i=1}^{m} d_i \lambda_i - \sum_{j=1}^{n} \sum_{i=1}^{m} d_i \overline{y}_j v_{ij} - \sum_{j=1}^{n} s_j \overline{y}_j \mu_j$$

$$\text{s.t. } \lambda_i - v_{ij} - \mu_j \leq c_{ij} \qquad \forall i = 1, \ldots, m, j = 1, \ldots, n,$$

$$v_{ij} \geq 0, \ \mu_j \geq 0 \qquad \forall i = 1, \ldots, m, j = 1, \ldots, n.$$

The full master problem is

$$\min \sum_{j=1}^{n} f_j y_j + \theta \tag{5.18}$$

$$\text{s.t. } \sum_{j=1}^{n} s_j y_j \geq \sum_{i=1}^{m} d_i, \tag{5.19}$$

$$\sum_{i=1}^{m} d_i \lambda_i^p - (\sum_{j=1}^{n} \sum_{i=1}^{m} d_i v_{ij}^p + \sum_{j=1}^{n} s_j \mu_j^p) y_j \leq \theta \qquad \forall p \in H^P, \tag{5.20}$$

$$y_j \in \{0, 1\} \ \forall i = 1, \ldots, m, j = 1, \ldots, n. \tag{5.21}$$

H^P denote the set of the extreme points. Constraints (5.19) are redundant and are added to the full master problem to strengthen the formulation. Note the absence of feasibility cuts as the primal subproblem is always feasible [75].

The performance of ABBC is tested by solving 35 instances of CFLP from OR Library (http://people.brunel.ac.uk/ mastjjb/jeb/info.html). The Following results are reported

in Table 5.1:

$Name$:	Instance Name.
n :	Number of Facilities.
m :	Number of Customer Locations.
$Nodes$:	Number of nodes of the search tree.
$Cuts$:	Number of Benders cuts that are generated.
$Cuts0$:	Number of Benders cuts that are generated at the root node.
$CutsAVG$:	Average number of Benders cuts per node, other than the root node.
$CPUAC$:	Total computational time in seconds spent on calculating the analytic center.
CPU :	Total computational time in seconds.
Sol :	Optimal Solution.

When comparing ABBC to KBBC, we observe that on average ABBC and KBBC explore the same number of nodes. With the exception of CAP VIII-1, CAP VIII-2, CAP IX-3, and CAP XI-3 where ABBC explores more nodes than KBBC, both branch-and-cut algorithms explore exactly the same number of nodes. ABBC might explore more nodes than KBBC because ABBC generates fewer Benders cuts than KBBC therefore nodes might be fathomed earlier in KBBC. Furthermore, the extra number of nodes that are explored in ABBC is not significant. In CAP VIII-1, 4 extra nodes are explored while in CAP VIII-2, CAP IX-3, and CAP XI-3 only 2 extra nodes are explored. For all 35 instances, ABBC required fewer Benders cuts to find the optimal solution than KBBC, generating on average 3 times less cuts. This confirms that the Benders cuts that are generated from the analytic center of the master problem are better than cuts that are generated from the extreme points as in KBBC. Having fewer and better cuts improves the computational performance since the problems that are solved are relatively smaller. ABBC found the optimal solution faster than KBBC for the majority of the tested instances and on average was 2.5 times faster than KBBC. In ABBC, on average 19% of the computational time is

				ABBC					KBBC				Benders			Benders with Pareto Cuts			
	Name	n	m	Nodes	Cuts	Cuts0	Cuts AVG	CPU AC	CPU	Nodes	Cuts	Cuts0	Cuts AVG	CPU	Nodes	Cuts	CPU	Nodes	Cuts	CPU	Sol
1	CAP IV-2	16	50	5	30	9	5.25	0.17	1.13	5	56	29	6.75	1.72	41768	16	29.19	11493	5	7.95	1098
2	CAP IV-3	16	50	7	39	12	4.5	0.25	1.52	7	62	24	6.33	1.84	22910	28	17.83	11513	5	8.05	1153
3	CAP IV-4	16	50	7	40	15	4.17	0.26	1.65	7	68	27	6.83	2.13	43413	27	32.26	11549	5	7.87	1235.5
4	CAP V-1	16	50	9	50	15	4.38	0.25	1.72	9	159	66	11.63	4.66	22246	34	19.06	14966	10	10.76	1025. 21
5	CAP VI-1	16	50	1	10	10	-	0.04	0.32	1	28	28	-	0.63	7213	27	7.97	2371	5	2.28	932.62
6	CAP VI-2	16	50	1	10	10	-	0.04	0.34	1	29	29	-	0.71	15278	38	16.74	2374	6	2.01	977.8
7	CAP VI-3	16	50	9	51	14	4.63	0.23	1.65	9	110	56	6.75	2.62	10935	39	13.19	2632	10	2.66	1014.06
8	CAP VI-4	16	50	1	13	13	-	0.06	0.48	1	49	49	-	1.16	4792	22	5.6	2761	13	3.17	1045.65
9	CAP VII-1	16	50	1	13	13	-	0.04	0.37	1	20	20	-	0.42	5238	26	6.58	30	4	0.41	932.62
10	CAP VII-2	16	50	1	11	11	-	0.06	0.38	1	23	23	-	0.45	10830	32	12.08	31	5	0.56	977.8
11	CAP VII-3	16	50	1	13	13	-	0.08	0.38	1	26	26	-	0.54	6503	31	8.51	163	9	1.07	1010.64
12	CAP VII-4	16	50	1	11	11	-	0.07	0.36	1	10	10	-	0.22	2823	23	4.27	59	5	0.57	1034.98
13	CAP VIII-1	25	50	7	53	19	5.67	0.78	3.19	3	169	152	8.5	8.81	*	*	*	*	*	*	838.5
14	CAP VIII-2	25	50	27	153	24	4.96	2.27	11.39	25	520	184	14	37.7	*	*	*	*	*	*	910.89
15	CAP VIII-3	25	50	31	166	38	4.27	2.94	12.94	31	583	242	11.37	44.74	*	*	*	*	*	*	975.89
16	CAP VIII-4	25	50	171	681	40	3.77	18.51	90.14	171	2641	249	14.07	582.83	*	*	*	*	*	*	1069.37
17	CAP IX-1	25	50	1	15	15	-	0.13	0.69	1	56	56	-	1.85	*	*	*	11318	10	9.26	796.65
18	CAP IX-2	25	50	5	38	22	4	0.44	1.8	5	83	47	9	2.69	*	*	*	12718	18	11.93	855.73
19	CAP IX-3	25	50	25	103	18	3.54	0.94	5.1	23	292	52	10.91	12.62	*	*	*	26872	44	33.55	896.62
20	CAP IX-4	25	50	29	131	19	4	1.58	7.55	29	233	67	5.93	9.44	*	*	*	23135	47	32.48	946.05
21	CAP X-1	25	50	1	11	11	-	0.09	0.56	1	53	53	-	1.82	3083958	126	5545.7	888	12	3.23	796.65
22	CAP X-2	25	50	1	15	15	-	0.09	0.71	1	35	35	-	0.98	682413	101	1113.06	1159	9	2.74	854.7
23	CAP X-3	25	50	1	17	17	-	0.11	0.82	1	29	29	-	0.77	12687	29	17.73	1614	14	4.25	893.78
24	CAP X-4	25	50	1	14	14	-	0.13	0.72	1	17	17	-	0.56	2437	13	4.92	830	10	2.78	928.94
25	CAP XI-1	50	50	5	41	22	4.75	0.89	6.02	5	450	375	18.75	65.72	*	*	*	*	*	*	826.12
26	CAP XI-2	50	50	15	129	35	6.71	3.23	24.66	15	590	240	25	88.26	*	*	*	*	*	*	901.38
27	CAP XI-3	50	50	47	335	36	6.5	15.69	105.69	45	1217	558	14.98	272.99	*	*	*	*	*	*	970.57
28	CAP XII-1	50	50	1	18	18	-	0.36	2.32	1	88	88	-	5.97	*	*	*	270518	26	224.35	793.44
29	CAP XII-2	50	50	7	54	24	5	0.97	6.56	7	130	80	8.33	8.79	*	*	*	384116	56	434.16	852.52
30	CAP XII-3	50	50	17	109	31	4.88	1.87	14.61	17	222	90	8.25	16.64	*	*	*	415341	69	500	895.3
31	CAP XII-4	50	50	45	364	27	7.66	21.06	88.76	45	417	110	6.98	39.78	*	*	*	264866	44	266.59	946.05
32	CAP XIII-1	50	50	1	18	18	-	0.45	2.5	1	66	66	-	4.27	*	*	*	42547	23	60.99	793.44
33	CAP XIII-2	50	50	1	18	18	-	0.43	2.54	1	52	52	-	3.3	*	*	*	14960	22	29.58	851.5
34	CAP XIII-3	50	50	1	20	20	-	0.42	2.86	1	36	36	-	2.31	*	*	*	3789	19	16.64	893.08
35	CAP XIII-4	50	50	1	17	17	-	0.52	2.56	1	19	19	-	1.36	*	*	*	1899	9	7.68	928.94
	Average			13.86	80.31	18.97	4.92	2.16	11.57	13.57	247.09	94.11	10.8	35.23	248465.25	38.25	428.42	54875.43	18.36	60.27	

-: Branch-and-cut did not go past node 0.

∗ : Instance could not be solved because the solver ran out of memory.

Table 5.1: Computational Results for the Capacitated Facility Location Problem

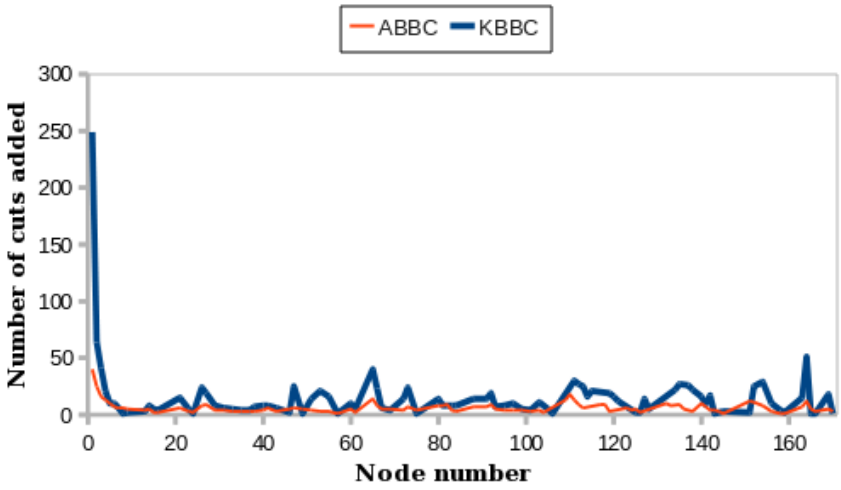

Figure 5.1: Number of cuts added at each node of the branch-and-cut (CAP VIII-4)

spent on calculating the analytic center. In both ABBC and KBBC, a large number of cuts is first generated early in the branch-and-cut tree while fewer cuts are generated later. On average, ABBC generated 2.85 times more cuts at node 0 than the remaining nodes. KBBC generates 2.63 times more cuts at node 0 than the remaining nodes. Figure 5.1 shows the number of cuts that are generated at each of the 171 nodes that are explored when instance CAP VIII-4 is solved. When ABBC is applied, 40 cuts are generated at node 0 while on average 3.77 cuts are generated at the remaining nodes. When KBBC is applied, 249 cuts are generated at node 0 while on average 14.07 cuts are generated at the remaining nodes.

Evaluating the effect of the pareto-optimal cuts on the classical iterative implementation of Benders decomposition, we observe that the pareto-optimal cuts significantly improve the performance of the classical Benders decomposition. The pareto-optimal cuts are of better quality than the classical cuts and fewer cuts are generated when the pareto-optimal cuts are used. We note that at each iteration of the classical Benders, a single cut is generated and therefore the number of cuts indicates the number of iterations which also indicates the number of times the integer master problem is solved. When the classical Benders cuts

are used in the classical Benders decomposition, 16 problems were solved while 19 other problems failed to solve because the algorithm ran out of memory. When pareto-optimal cuts are used, 28 problems were solved while 7 other problems failed to solve. The addition of pareto-optimal cuts improved the performance of the classical Benders decomposition in all the tested instances.

Comparing ABBC to the classical Benders decomposition with pareto-optimal cuts, we observe that ABBC outperformed the classical Benders decomposition in all the instances in terms of computational time and in terms of number of nodes. However, on average the pareto-optimal cuts that were generated were 2.44 times less than the Benders cuts that were generated in ABBC. This is due to the fact that in classical Benders decomposition, Benders cuts are generated only when the optimal solution of the master problem is solved while ABBC generates cuts at every node of the tree and hence ABBC generates more cuts than the classical Benders decomposition. However solving the integer master problem repeatedly such as in classical Benders decomposition has a big disadvantage which is revealed in terms of the number of nodes that are explored which is significantly higher than the number of nodes explored in ABBC. For the instances that were solved, classical Benders explored 8442 times more nodes than ABBC. Furthermore, the computational time of ABBC is also significantly lower than that of the classical Benders decomposition. For the instances that were successfully solved, the computational time of ABBC was always better than the computational time of classical Benders decomposition and on average 11 times faster.

The computational results showed that Benders-branch-and-cut has a large computational advantage over the classical Benders decomposition both with and without pareto-optimal cuts. While ABBC and KBBC both explore comparable number of nodes, ABBC requires fewer cuts and computational time to solve each node. The advantage that ABBC has is due to the fact that the Benders cuts that are generated from the analytic center seem to be better than the Benders cuts that are generated from the extreme points.

5.5.2 The Multicommodity Capacitated Fixed Charge Network Design Problem

To further confirm our computational findings, we test ABBC and KBBC by solving 30 instances of the multicommodity capacitated fixed charge network design problem (MCFND) that was successfully solved by Benders decomposition [17]. MCFND is described as follows. Given a directed graph $G = (N, A)$ with N nodes and A arcs. Each arc (i, j) has a capacity q_{ij}. A set K of commodities should be shipped between origin and destination pairs. For each commodity k, an amount w^k should be shipped from an origin node $O(k)$ to a destination node $D(k)$. The objective is to minimize the network cost which include 2 kinds of costs: the cost c_{ij}^k of shipping one unit of commodity k on arc (i, j), and the fixed cost f_{ij} which is incurred if arc (i, j) is used in the network. The binary variable y_{ij} associated with arc (i, j) takes the value of 1 if arc (i, j) is used in the network and 0 otherwise. The continuous variable x_{ij}^k denotes the amount of flow of commodity k on arc (i, j). A formulation of the multicommodity capacitated fixed charge network design problem as given in [17]

$$
\begin{aligned}
&\min \sum_{k\in K}\sum_{(i,j)\in A} c_{ij}^k x_{ij}^k + \sum_{(i,j)\in A} f_{ij} y_{ij}, \\
&\text{s.t.} \sum_{j\in N^+(i)} x_{ij}^k - \sum_{j\in N^-(i)} x_{ij}^k = \begin{cases} w^k & \text{if } i = O(k) \\ -w^k & \text{if } i = D(k) \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in N,\ \forall k \in K, \\
&\sum_{k\in K} x_{ij}^k \le q_{ij} y_{ij} \quad \forall (i,j) \in A, \\
&x_{ij}^k \ge 0 \quad \forall (i,j) \in A,\ \forall k \in K, \\
&y_{ij} \in \{0,1\} \quad \forall (i,j) \in A,
\end{aligned}
$$

where $N^+(i) := \{j \in N : (i,j) \in A\}$ and $N^-(i) := \{j \in N : (j,i) \in A\}$. Fixing the variables $y = \overline{y}$ leads to the following Benders primal subproblem

$$\sum_{(i,j)\in A} f_{ij}\overline{y}_{ij} + \min \sum_{k\in K}\sum_{(i,j)\in A} c_{ij}^k x_{ij}^k, \tag{5.22}$$

$$\text{s.t.} \sum_{j\in N^+(i)} x_{ij}^k - \sum_{j\in N^-(i)} x_{ij}^k = \begin{cases} w^k & \text{if } i = D(k) \\ -w^k & \text{if } i = D(k) \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in N,\ \forall k \in K, \tag{5.23}$$

$$\sum_{k\in K} x_{ij}^k \le q_{ij}\overline{y}_{ij} \quad \forall (i,j) \in A, \tag{5.24}$$

$$x_{ij}^k \ge 0 \quad \forall (i,j) \in A,\ \forall k \in K. \tag{5.25}$$

Let μ_{ki} be the dual variable of constraint (5.23), and λ_{ij} be the dual variable of constraint (5.24), the full master problem is

$$\min \sum_{(i,j)\in A} f_{ij}y_{ij} + \theta, \tag{5.26}$$

$$\text{s.t.} \sum_{(i,j)\in A} \lambda_{ij}^p q_{ij} y_{ij} + \sum_{k\in K} d_k \mu_{kO(k)}^p - d_k \mu_{kD(k)}^p \le \theta \quad \forall p \in H^P, \tag{5.27}$$

$$\sum_{(i,j)\in A} \lambda_{ij}^r q_{ij} y_{ij} + \sum_{k\in K} d_k \mu_{kO(k)}^r - d_k \mu_{kD(k)}^r \le 0 \quad \forall r \in H^R, \tag{5.28}$$

$$y_{ij} \in \{0,1\} \quad \forall (i,j) \in A. \tag{5.29}$$

H^P and H^R denote the sets of the extreme points and extreme rays of the dual subproblem.

We evaluate the performance of ABBC and KBBC by solving 30 instances of the MCFND problem. The instances are generated using the mulgen generator [19]. The computational results are reported in Table 7.1. The name of each problem indicates the size of the

considered network. For instance, $p\_n\_m\_k$ denotes a network with n nodes, m arcs, and k commodities. Networks with 10 and 15 nodes are considered while the number of arcs is varied between 30, 40, 50, 60, and 70. Additionally, the number of commodities is varied between 2, 3 and 4.

In Table 7.1, the results of ABBC and KBBC are reported while the evaluation with classical Benders with and without pareto-optimal cuts have been omitted because many of the CFLP instances failed to solve within the time and memory limitations. In our testing only $p\_10\_30\_2$, $p\_10\_40\_2$, $p\_15\_30\_2$, $p\_15\_30\_2$ could be solved without running out of memory. Furthermore, from the CFLP computational testing, both ABBC and KBBC significantly outperformed both Benders algorithms.

Comparing ABBC to KBBC, we observe that ABBC tends to explore more nodes than KBBC. In 14 instances ABBC and KBBC explored the same number of nodes while in the remaining 16 instances ABBC explored more nodes. On average ABBC required 80.47 nodes while KBBC required 67.6 nodes. The significant difference of ABBC and KBBC is in the quality of the cuts that each method generates. KBBC generated on average 2472.3 cuts while ABBC generated 1107.67 cuts only, hence confirming that ABBC generates better cuts than KBBC. With fewer and better cuts, ABBC outperforms KBBC in terms of computational time. On average, ABBC was 4 times faster than KBBC. For $p\_10\_70\_4$, ABBC generated 5 times less cuts than KBBC and was 4 times faster. On average, 24% of the computational time of ABBC is spent on calculating the analytic center.

5.6 Conclusion

Even with the addition of pareto-optimal cuts, classical Benders decomposition suffers from computational inefficiency since the master problem which is a mixed integer problem is solved repeatedly. We propose a branch-and-cut based Benders decomposition algorithm. Rather than resolving the mixed integer master problem from scratch each time a Benders

		ABBC						KBBC					
	Name	Nodes	Cuts	Cuts0	Cuts AVG	CPU AC	CPU	Nodes	Cuts	Cuts0	Cuts AVG	CPU	Sol
1	p_10_30_2	19	75	23	2.89	0.07	0.47	19	80	37	4.44	0.53	18721
2	p_10_40_2	23	145	35	6.59	0.27	1.59	21	152	59	7.6	1.72	16983
3	p_10_50_2	33	164	29	5.12	1.36	6.05	33	711	253	22.22	40.86	9688
4	p_10_60_2	23	119	33	5.41	1.51	5.92	23	448	162	20.36	21.01	21902
5	p_10_70_2	17	95	27	5.94	1.21	6.12	17	576	245	36	38.99	15958
6	p_10_30_3	63	338	35	5.45	1.55	8.38	57	450	184	8.04	10.27	22281
7	p_10_40_3	53	356	37	6.85	2.97	12.07	53	602	250	11.58	25.47	20220
8	p_10_50_3	45	511	43	11.61	4.45	29.53	35	1427	782	41.97	183.1	13970
9	p_10_60_3	29	172	39	6.14	2.46	13.37	25	1733	592	72.21	321.07	24129
10	p_10_70_3	73	935	41	12.99	30.21	129.99	73	1657	480	23.01	377.59	21690
11	p_10_30_4	61	552	50	9.2	3.51	21.09	41	938	413	23.45	47.52	29177
12	p_10_40_4	233	2729	50	11.76	102.92	742.89	209	3081	823	14.81	802.71	27995
13	p_10_50_4	121	2016	50	16.8	96.54	454.63	93	2513	802	27.32	646.36	31219
14	p_10_60_4	329	5210	41	15.88	1026.89	4067.88	293	9670	4239	33.12	12539.72	29048
15	p_10_70_4	91	2567	33	28.52	182.84	1135.31	91	13239	10053	147.1	22925.32	25086
16	p_15_30_2	25	57	27	1.25	0.07	0.32	21	76	24	3.8	0.52	41599
17	p_15_40_2	19	96	24	5.33	0.42	2.32	19	354	82	19.67	7.56	21257
18	p_15_50_2	21	105	29	5.25	0.88	3.75	21	830	371	41.5	55.76	22350
19	p_15_60_2	101	810	30	8.1	12.11	72.05	89	1390	439	15.8	179.92	31532
20	p_15_70_2	21	428	27	21.4	7.05	40.37	21	969	590	48.45	77.59	19204
21	p_15_30_3	33	164	37	5.12	0.27	1.58	31	175	68	5.83	1.97	23723
22	p_15_40_3	71	1265	46	18.07	18.61	133.45	59	2405	1421	41.47	345.79	32400
23	p_15_50_3	27	152	22	5.85	0.85	5.6	27	961	515	36.96	76.07	36347
24	p_15_60_3	471	4945	66	10.52	1051.9	4135.12	311	5648	1230	18.22	4715.25	28476
25	p_15_70_3	151	4159	47	27.73	952.98	3731.07	93	10751	7883	116.86	17217.22	27557
26	p_15_30_4	45	470	40	10.68	3.17	14.85	39	668	378	17.58	19.19	67389
27	p_15_40_4	65	1165	53	18.2	27.91	106.06	65	2813	897	43.95	603.37	47009
28	p_15_50_4	67	1413	52	21.41	47.98	238.03	65	2593	717	40.52	736.88	54074
29	p_15_60_4	51	1405	48	28.1	51.48	277.69	51	2984	1023	59.68	1191.19	43115
30	p_15_70_4	33	612	48	19.12	11.71	59.91	33	4275	1832	133.59	3343.14	32781
	Average	80.47	1107.67	38.73	11.91	121.54	515.25	67.6	2472.3	1228.13	37.9	2218.46	

Table 5.2: Computational Results for the Multicommodity Capacitated Fixed Charge Network Design Problem

cut is generated, the advantage of branch-and-cut is that the Benders cuts are generated dynamically as the master problem is solved. We show that the Benders cuts that are generated are global cuts and are used at all the nodes of the branch-and-cut tree. Furthermore, ACCPM is used to generate tight cuts and improve the computational performance. We show that the Benders cuts that are generated using the analytic center of the master problem are pareto-optimal cuts. The computational results confirm the Benders cuts that are generated using ACCPM are tighter than the Benders cuts that are generated using Kelley's cutting plane method.

ABBC is compared to KBBC, to classical Benders decomposition, and to classical Benders decomposition with pareto-optimal cuts. Computational results on the capacitated facility location problem and the multicommodity capacitated fixed charge network design problem showed that ABBC was always better than KBBC. Both ABBC and KBBC significantly outperformed the classical Benders decomposition algorithm and the classical Benders algorithm with pareto-optimal cuts.

Chapter 6

Using ACCPM to Find Integer Feasible Solutions

[1] Finding feasible solutions for mixed integer programming (MIP) is a hard problem that is important in practice. Additionally, finding feasible solutions for MIP is an important step towards finding optimal solutions. Good feasible solutions help in fathoming branches earlier in the branch-and-bound tree and contribute to the reduction of the computational time and the memory required.

The literature is rich with heuristics for MIP. Hillier [40] was among the first to propose a heuristic based on interior point methods. His three-phase method starts by identifying an interior path connecting an interior point and the optimal solution of the LP relaxation. In Phase 2, a search around the interior path is conducted to find a feasible integer solution, but with no guarantees for a successful termination. In Phase 3, the algorithm attempts to find other feasible integer solutions that improve the objective function value. Hillier's algorithm was implemented within a branch-and-bound algorithm in [43].

[1] The material in this chapter has been published in J. Naoum-Sawaya and S. Elhedhli, "An Interior Point Cutting Plane Heuristic for Mixed Integer Programming", Computers and Operations Research, 38(9): 1335-1341 (2011).

Balas and Martin [6] use the fact that every 0-1 binary problem is equivalent to a linear problem with all slack variables being basic. Their proposed heuristic solves the linear programming relaxation and pivots all the slack variables to the basis. The extension of [6] to general mixed integer problems is presented in [8]. Saltzman and Hillier [67] describe an algorithm that enumerates feasible 1-ceiling points around the optimal linear programming solution. Glover and Laguna [32, 33] propose a heuristic framework based on cut search procedures for general mixed integer problems. Løkketangen and Glover [51] present a tabu search heuristic while Balas et al. [7] propose an algorithm that enumerates extended facets of the octahedron to solve 0-1 binary problems.

The feasibility pump for 0-1 binary problems was introduced in Fischetti et al. [27] and was extended to general mixed integer problems in [12]. In Achterberg and Berthold [2], a modification of the feasibility pump that improves the quality of the feasible solutions is presented. The feasibility pump heuristic iteratively solves the linear programming (LP) relaxation of the MIP problem to generate a point y^* which is rounded to the nearest integer point $\tilde{y}$. The point y^* satisfies the linear constraints, while $\tilde{y}$ satisfies the integrality constraints. A feasible integer solution is found when the two points coincide. If the same points are generated repeatedly, random perturbations that randomly shift the values of $\tilde{y}$ up and down are performed to restart the search at a different integer point. If after a fixed number of iterations a feasible integer solution is not found, an integer program is solved to find the closest integer point to y^*; by solving $\min |y - y^*|$ subject to the constraints of the original MIP where y^* is the last found LP solution.

Compared to extreme points, the motivation behind the use of interior points resides in the fact that it is more likely that rounding an interior point to the nearest integer will result in a feasible integer solution. The geometric center of the convex set appears to be the best candidate for an interior point where its rounding is feasible. However, finding the geometric center is a hard problem. The analytic center [69] often lies close to the geometric center and is easier to calculate. In addition, the location of the analytic center

can be displaced by modifying the weights on the corresponding constraints [34]. Our approach uses weights to guide the analytic center towards regions where rounding will likely give a feasible integer solution.

6.1 The Analytic Center Feasibility Method

The analytic center feasibility method (ACFM) is implemented in three phases. In Phase-I, two line segments, each connecting one extreme point to the continuous analytic center, are identified. The two extreme points are the points that maximize and minimize the objective function value of the LP relaxation, respectively. Candidate integer solutions are found by rounding the solutions on the line segments to the nearest integer. If none of the identified integer solutions is feasible, then the weights of the constraints are updated in Phase-II to shift the analytic center to a new position and restart the search. If after a predetermined number of iterations an integer feasible solution is not found, then an enumeration stage is executed in Phase-III.

6.1.1 Phase-I

To provide a detailed description, let us introduce the generic mixed integer problem:

$$\begin{aligned} &\min\ b^T y \\ &\text{s.t. } A^T y \leq c, \\ &\qquad y_j \text{ integer } \forall j \in J. \end{aligned} \tag{6.1}$$

In Phase-I, a search region is identified and a simple search for integer feasible solutions is conducted. The search region is formed of two line segments. Each line segment connects one extreme point to the weighted analytic center. Given the LP relaxation of problem

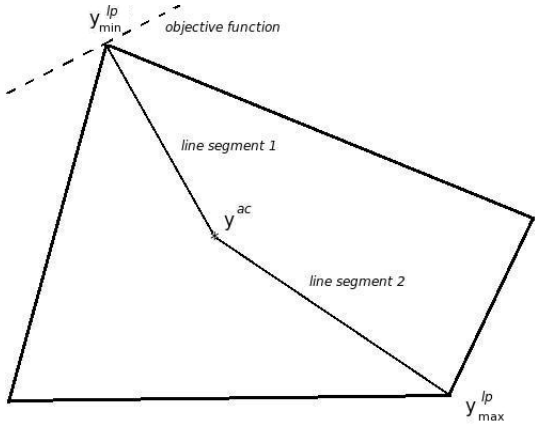

Figure 6.1: Search Space

(7.1), the first extreme point is the optimal solution $y^{lp}_{\min}$ of $\{\min b^T y : A^T y \leq c\}$ while the second extreme point is the optimal solution $y^{lp}_{\max}$ of $\{\max b^T y : A^T y \leq c\}$. The interior point is the weighted analytic center y^{ac} of $\mathcal{F} = \{y : A^T y \leq c, b^T y \leq z_u\}$ where z_u is an upper bound on the objective. As shown in Figure 1, the search region is the two line segments:

$$y = \alpha y^{ac} + (1-\alpha) y^{lp}_{\min}, \quad 0 \leq \alpha \leq 1, \tag{6.2}$$

$$y = \alpha y^{ac} + (1-\alpha) y^{lp}_{\max}, \quad 0 \leq \alpha \leq 1. \tag{6.3}$$

The search is performed on a selected set of points corresponding to α values that are decremented in steps. Since y^{ac} lies at the center of the polyhedron, it is expected that feasible integer solutions are found near y^{ac}. So starting from a value of 1, α is gradually decreased by 0.05 and the nearest rounded integer solution is found (the search starts at y^{ac} and moves towards $y^{lp}_{\min}$ and $y^{lp}_{\max}$). The algorithm is stopped when $\alpha = 0$ and the best integer solution is selected.

Let $\overline{y}$ be a point belonging to the search region, the integer point $\bar{y}_I$ that is closest to $\bar{y}$ is found by first rounding $\bar{y}_j$, $j \in J$ to the nearest integer point $[\bar{y}]$. Then, the continuous

components are found by solving the following linear problem

$$\begin{aligned} &\min\ b^T y \\ &\text{s.t. } A^T y \leq c, \\ &\quad y_j = [\bar{y}_j] \quad \forall j \in J. \end{aligned} \tag{6.4}$$

If problem (6.4) is feasible, then its optimal solution y_I is a feasible integer point, otherwise the nearest integer solution is infeasible, and the search continues.

6.1.2 Phase-II

Phase-II reshapes the polyhedron to either improve the quality of the integer feasible solution (if one was found in Phase-I) or reshapes the polyhedron to find an integer feasible solution (if none was found in Phase-I). Through the definition of the localization set $\mathcal{F}$, the quality of the solution that is found in Phase-I is bounded by the value of z_u which is an upper bound on the resulting feasible solution. If a feasible solution was found in Phase-I but it is desired to search for a better quality solution, the bound is updated to $z_u = b^T y_I$ and Phase-I is rerun.

On the other hand, if no feasible integer solution was found in Phase-I, the weighted analytic center y^{ac} around which it is very likely to find an integer feasible solution is taken as a reference point to reshape the polyhedron $\mathcal{F}$. Since no feasible solution was found in Phase-I then $y_I = [y^{ac}]$ is not in the interior of $\mathcal{F}$, so y_I should be pushed towards the interior of $\mathcal{F}$. This can be done by changing the position of y^{ac}. As detailed in Section 2.4.1, increasing the weight of a constraint will push the analytic center away from it. To move y_I towards the interior of $\mathcal{F}$, the weights of the violated constraints are increased and the continuous analytic center is recomputed by rerunning Phase-I. The new weights will force the analytic center away from the violated constraint, increasing the possibility of finding feasible integer solutions.

Numerical tests on mixed integer linear problems showed that the process of pushing the integer points towards feasibility is slow. This is due to the fact that the constraints are formed of integer and continuous variables. By incrementing the weights, the continuous components of the analytic center will shift, thus reducing the change in the integer components. Therefore, despite the change in the position of the analytic center, the rounded solution will not change and will remain infeasible. To force the values of the integer components of the analytic center to change, new cuts are added to the initial problem. For each constraint

$$\sum_j a_j y_j + \sum_i a_i y_i \leq c_m, \tag{6.5}$$

that is violated by $y_I = [y^{ac}]$, i.e.

$$\sum_j a_j [y_j^{ac}] + \sum_i a_i y_i^{ac} > c_m,$$

where the variables y_j are the integer variables and y_i are the continuous variables, the following cut is added

$$\sum_j a_j y_j \leq c_m - \sum_i a_i y_i^{ac}. \tag{6.6}$$

Then instead of incrementing the weight of the violated constraint (6.5), only the weight of the new constraint (6.6) is incremented. To avoid adding a large number of cuts if the same constraint (6.5) is violated in multiple iterations and making the problem harder to solve, we append constraint (6.6) the first time constraint (6.5) is violated and then in subsequent iterations, we only change the right hand side of the previously added constraint (6.6) and increment the associated weight. We note that constraint (6.6) is may cut off feasible integral solutions. The aim with the cut is to focus the solution approach on finding one

good feasible solution, so cutting off some integer feasible solutions can be regarded as making a trade-off between speeding up the search and finding the best possible solution. In addition to the constraint generation, computational testing showed that a cut formed by a convex combination of all violated constraints helps in guiding the analytic center towards feasible integer solutions. For each constraint i, a coefficient l_i is assigned and initialized to zero. Each time Phase-I does not find a feasible integer solution, the coefficient of each violated constraint is incremented. The weighted sum $W(y)$ is given by $\sum_i l_i a^i(y)$, where $a^i(y)$ denotes the left-hand-side of constraint i and l_i its corresponding weight. $W(y)$ is then evaluated at the analytic center y^{ac} and a cut of the form $W(y) \leq W(y^{ac})$ is added to the polyhedron. For example, given the analytic center $y_1^{ac} = 0.6$ and $y_2^{ac} = 2.6$, the rounded point $[y_1^{ac}] = 1$ and $[y_2^{ac}] = 3$ violates the two constraints $y_1 + 3y_2 \leq 9$ and $3y_1 + 5y_2 \leq 16$. Assuming that the two constraints have weights $l_1 = 2$ and $l_2 = 3$ respectively, the added cut is $W(y) = 2(y_1 + 3y_2) + 3(3y_1 + 5y_2) = 11y_1 + 21y_2 \leq 61.2$ which goes through the current analytic center. We note that in our implementation, $W(y)$ is updated at every iteration and the corresponding cut is modified accordingly and therefore it is a single cut that is iteratively updated rather than multiple cuts that are being generated.

The iteration between Phase-I and Phase-II may run indefinitely with no guarantees of finding an integer feasible solution. This step can be stopped when an iteration limit, a time limit, or a feasible solution with a desired quality is reached. In our testing, we use an iteration limit of 20 as a stopping criterion.

6.1.3 Phase-III

Similar to the approach adopted in [12], if after several iterations between Phase-I and Phase-II a feasible solution is still not found, then the following problem is solved

$$\begin{aligned} &\min\ d(y, y^{ac}) \\ &\text{s.t. } A^T y \leq c, \\ &\qquad y_j \text{ integer} \quad \forall j \in J, \end{aligned} \tag{6.7}$$

with y^{ac} being the analytic center found in the last iteration. The solver is stopped after the first feasible solution is found. As it is expected that y^{ac} will have a feasible integer solution in its vicinity, a feasible solution might be found in a relatively short amount of time.

The ACFM algorithm is outlined next.

1. Initialization: Find $y^{lp}_{\min}$ and $y^{lp}_{\max}$.
2. While the stopping criterion are not satisfied

Phase I:

- 2.1 Initialize $\alpha = 1$
- 2.2 Calculate the analytic center y^{ac}
- 2.3 While $\alpha > 0$
  - 2.3.1 Find y from (6.2)
  - 2.3.2 Find the nearest integer point $[y]$
  - 2.3.3 Check if $[y]$ is feasible by solving (6.4)
  - 2.3.4 Decrease α by 0.05
- 2.4 End While
- 2.5 Initialize $\alpha = 1$
- 2.6 While $\alpha > 0$
  - 2.6.1 Find y from (6.3)
  - 2.6.2 Find the nearest integer point $[y]$
  - 2.6.3 Check if $[y]$ is feasible by solving (6.4)
  - 2.6.4 Decrease α by 0.05
- 2.7 End While

Phase II:

- 2.8 If a feasible solution y^I was found in Phase-I update the bound $z_u = b^T y_I$
  - 2.8.1 Exit Phase-II
- 2.9 Else
  - 2.9.1 For each violated constraint
    - 2.9.1.1 Add a cut of the form (6.6) if it wasn't added in a previous iteration
    - 2.9.1.2 Increment the weight of the violated constraint
    - 2.9.1.3 Increment the coefficient of the violated constraint
  - 2.9.2 Compute the weighed cut $Wy \leq Wy^{ac}$

2.9.3 Update the constraints with the new cuts

3. End While

Phase III:

4. If no feasible integer solution was found

4.1 Solve problem (6.7) with $y^* = y^{ac}$

4.2 Stop after the first feasible solution is found

6.2 Problems with Equality Constraints

Analytic centers can only be computed for polytopes with relative interior. The previous discussion focused on problems with inequality constraints. In this section, we consider the extension of ACFM to mixed integer problems formed by inequality as well as equality constraints. The idea is to transform equality constraints to inequalities by removing one of the non-negative variables. For each constraint of the form $a_1x_1 + a_2x_2 = b$ where x_2 is either a continuous or integer variable that is always nonnegative and $a_2 > 0$, the inequality $\frac{a_1}{a_2}x_1 \leq \frac{b}{a_2}$ is valid. In our implementation, the continuous variable with the biggest coefficient is removed. If all the variables are discrete, then the discrete variable with the biggest coefficient is removed. Removing the variable with the biggest coefficient will most likely create a relatively large interior. After modifying all the equality constraints, the interior search algorithm is applied with the following changes. In Phase-I, the rounding to the nearest integer is done in two stages. First, the integer variables that are not removed from any of the equality constraints are rounded to the nearest integer (i.e. $y_j = [\bar{y}_j]\ \forall j \in J$, and y_j was not removed). Then, problem (6.4) is solved with those variables fixed, and the remaining integer variables relaxed to being continuous. If the problem is infeasible, Phase-I proceeds with no additional changes. If the problem is feasible, then all the variables that are supposed to be integer are rounded to the nearest integer using the optimal solution of problem (6.4). Problem (6.4) is then resolved with the new fixed variables and Phase-I proceeds with no additional changes. We note that problem (6.4) is always solved with the

original equality constraints and not with the modified constraints. If a feasible solution is found in Phase-I or if y^{ac} does not violate any of the equality constraints then the algorithm proceeds as it was previously described. However, if y^{ac} violates an equality constraint $A_{eq}y = c_{eq}$ such that $A_{eq}y^{ac} > c_{eq}$, then a cut of the form $A_{eq}y \leq A_{eq}y^{ac}$ is added. If y^{ac} violates an equality constraint $A_{eq}y = c_{eq}$ such that $A_{eq}y^{ac} < c_{eq}$, then a cut of the form $A_{eq}y \geq A_{eq}y^{ac}$ is added.

6.3 Fixing Infeasible Solutions

This section presents a modification to ACFM that aims at restoring the infeasibility of solutions at the end of Phase-II, instead of discarding them. Given an integer point y_I that is found to be infeasible, let A_S^T be the constraints that are violated by y_I, i.e. $A_S^T y_I > c$. The following mixed integer problem is then solved

$$\begin{aligned} &\min\ |y - y^{ac}| \\ &\text{s.t. } A_S^T y \leq c, \\ &\qquad y_j \text{ integer} \quad \forall j \in J. \end{aligned} \tag{6.8}$$

If A_S^T contains few constraints, and y^{ac} is a central point, then problem (6.8) is expected to be solved efficiently. Note that problem (6.8) is always feasible if problem (7.1) is feasible. Let y^S be the optimal solution of problem (6.8) and V be the index set of variables that have at least one non-zero coefficient in A_S^T, the following problem is then solved

$$\begin{aligned} &\min\ b^T y \\ &\text{s.t. } A^T y \leq c, \\ &\qquad y_j = y_j^S \quad \forall j \in V. \end{aligned} \tag{6.9}$$

Name	m	n	B	I	Card	Knap	Lin	Name	m	n	B	I	Card	Knap	Lin
mas74	13	151	150	0	1	0	12	neos-796608	286	311	0	119	23	0	263
mas76	12	151	150	0	1	0	11	neos-880324	348	261	232	0	14	0	334
misc07	212	260	259	0	140	0	72	neos-881765	278	712	712	0	278	0	0
noswot	182	128	75	25	7	0	175	neos-905856	403	686	686	0	403	0	0
pk1	45	86	55	0	0	0	45	neos-942886	359	464	448	0	271	0	88
pp08a	136	240	64	0	8	0	128	neos-1121679	6	62	50	0	0	6	0
pp08aCUTS	246	240	64	0	8	0	238	neos-1211578	356	260	130	0	151	75	130
rout	291	556	300	15	15	5	271	neos-1228986	356	260	130	0	151	75	130
vpm2	234	378	168	0	0	24	210	neos-1337489	356	260	130	0	151	75	130
neos2	1103	2101	1040	0	26	0	1077	neos-1420205	383	231	126	105	5	63	315
neos3	1442	2747	1360	0	34	0	1408	neos-1425699	89	105	5	80	59	0	30
neos5	63	63	53	0	63	0	0	neos-1430701	668	312	156	0	362	150	156
neos-501453	40	165	0	147	225	0	40	neos-1440447	561	260	130	0	306	125	130
neos-530627	113	103	0	28	25	0	88	neos-1460246	306	285	266	0	215	0	91
neos-584851	661	445	40	405	4	0	657	prod1	280	250	149	0	7	100	101
neos-595905	704	1200	312	0	40	0	664	prod2	211	301	200	0	10	100	101
neos-709469	469	224	224	0	212	86	171	rlp1	68	461	450	0	10	0	58
neos-717614	891	3049	84	2916	339	48	504	roy	162	149	50	0	2	0	160

Table 6.1: Test Problems

If problem (6.9) has an optimal solution y_S^*, then the nearest integer points to the line segment

$$y = \alpha y^{ac} + (1-\alpha) y_S^* \quad 0 \leq \alpha \leq 1,$$

are checked for feasibility. If a feasible integer solution y_I is found, then Phase-II is called to update the bounds. ACFM then proceeds as in Section 6.1.1.

6.4 Computational Results

This section presents computational results performed on 36 problems collected from MIPLIB [4] and CORAL [49]. The instances with the corresponding number of binary variables (B), the number of integer variables (I), the total number of variables (n), and the

total number of constraints (m) are described in Table 6.1. We also show the number of knapsack constraints (Knap) which are constraints with only positive coefficients, the number of cardinality constraint (Card) which are constraints involving a sum of variables only, and the number of unstructured constraints (Lin) where no particular structure is identified. We restrict the testbed to instances with at most 3000 variables and 3000 constraints as computing the analytic center becomes computationally intensive for large instances. The testing is aimed at evaluating the viability of the approach and the stability of the method. For large instances, an efficient implementation of the analytic center cutting plane method will be necessary. The computational testing was done on a Sunblade 2500 workstation with a 1.6 GHz processor and 2 Gb of RAM.

The different components of ACFM were coded in C, with Cplex 11.0 being used to solve the LP relaxation and problem (6.4) of Phase-I. All results are compared to the feasibility pump that is implemented on NEOS [21]. The analytic center based feasibility method and the feasibility pump are compared in terms of solution quality. Even though it was never called, Phase-III is set to run after 20 unsuccessful iterations between Phase-I and Phase-II. In Phase-III the setting of the MIP emphasis is set to favor feasibility over optimality.

ACFM is compared to the original feasibility pump (BFL-FP) introduced in [12] as well as the objective feasibility pump (AB-FP) described in [2]. The computational results are reported in Table 6.2. Columns (1) and (2) display the instance name and the number of iterations needed to reach the feasible solution displayed in Column (3), respectively. Column (4) indicates the percentage gap between the solution found by ACFM and the optimal solution. Column (5) displays the amount of CPU time in seconds spent on calculating the analytic centers, while Column (6) shows the total CPU time of the algorithm. Columns (7) and (9) display the solutions found using BFL-FP and AB-FP respectively, with the corresponding gaps displayed in Columns (8) and (10). The optimal solution is displayed in Column (10).

As detailed in Table 6.2, ACFM performed better than the feasibility pump in 22 instances including 4 instances where feasibility pump failed, found the same solution in 10 instances, and found a worse solution in 4 instances. For 9 instances, ACFM found the optimal solution. For all 36 instances, a feasible solution was found in less than 20 iterations and Phase-III was never called. In the worst case (neos-1440447), ACFM took 18 iterations to find a feasible integer solution. In terms of running times, the CPU time for both versions of the feasibility pump was not included in Table 6.2 due to the fact that all of the 32 instances where the feasibility pump was successful, the computational time was less than 1 second. The significant difference in computational time between ACFM and the feasibility pump is mainly due to the fact that the feasibility pump solves linear problems, while ACFM that relies on interior point principles, requires the calculation of analytic centers which is known to be computationally expensive. As shown in Column(5), almost 99% of the CPU time is spent on calculating the analytic centers. Having a computationally optimized method for calculating the analytic centers is essential to improve the performance of the algorithm. Finally although ACFM is computationally expensive, it found feasible solutions for 4 instances where the feasibility pump has failed thus showing the importance of searching in the interior to find feasible solutions. In particular, problems neos2, neos3, and neos-595905 contain very few knapsack and cardinality constraints while neos-717614 contains a large number of integer variables on top of that. ACFM is expected to be most successful in solving problems with general integer variables and no particular structure.

We also evaluate the effect of the cuts on the performance of ACFM by conducting computations without the cuts. The results are illustrated in Table 6.3. The results show that if cuts are not added, ACFM fails to find a feasible solution within 20 iterations for 9 of the 36 instances and performs more iterations for 14 instances. For the 27 instances for which a feasible solution is found, ACFM without cuts consumes on average 3.3 more iterations to find a feasible solution. For only one instance (prod1), a better feasible solution is found.

Finally, we evaluate the effect of enabling the infeasibility fixing procedure of Section 6.3. As shown in Table 6.4, fixing infeasibility leads to fewer iterations in 7 of the 32 instances. However in only 2 instances, a feasible solution is found faster. On average, the algorithm consumes additional 0.83 seconds when fixing infeasibility is enabled. For only one instance (neos-501474), a better feasible solution is found.

6.5 Conclusion

In this chapter, we presented an algorithm for finding feasible solutions for mixed integer problems based on analytic centers. A search is conducted around two line segments, connecting the analytic center to two extreme points of the LP relaxation of the MIP. Using weights, the analytic center is displaced iteratively and rounded in an attempt to lead to integer feasible points. Using 32 problems from MIPLIB and CORAL, the algorithm is tested and compared to the feasibility pump. The algorithm is found to outperform the feasibility pump in 32 of the 36 instances at the expense of taking longer computational time.

Future work will focus on reducing the computational time through a possibly better implementation. More importantly, the success of ACFM revitalizes the attention to the use of interior point approaches for feasibility in mixed integer non-linear.

	ACFM					BFL-FP		AB-FP		
Name	Iter	Obj	%Gap	AC time(s)	Total time(s)	Obj	%Gap	Obj	%Gap	Opt
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
mas74	7	**15026.47**	434.75	12.39	13.34	21923.62	680.2	19033.18	577.34	2810
mas76	1	**44877.42**	12.18	3.15	3.83	90213	125.5	50124	25.29	40005.1
misc07	13	4795	70.64	13.07	13.92	4795	70.64	**4600**	63.7	2810
noswot	3	-37	9.76	3.16	3.76	-29	29.27	**-39**	4.88	-41
pk1	1	**28.99**	163.55	1.08	1.13	108	881.82	57	418.18	11
pp08a	1	**9048.56**	23.11	2.25	3.1	13180	79.32	10810	47.07	7350
pp08aCUTS	1	**8458**	15.07	3.37	4.21	11070	50.61	8530	16.05	7350
rout	4	**1111.88**	3.18	150.87	152.93	1521.83	41.23	1294.17	20.1	1077.56
vpm2	6	**15.5**	12.73	40.97	42.65	16	16.36	16.5	20	13.75
neos2	3	**899.43**	97.74	261.41	277.34	-	-	-	-	454.86
neos3	2	**834.11**	126.14	174.19	185.77	-	-	-	-	368.84
neos5	1	**19**	26.67	0.35	0.35	**19**	26.67	25	66.67	15
neos-501474	6	**52211.89**	0.51	16.23	17.47	52290.5	0.67	52595.59	1.25	51944.7
neos-530627	1	**2995.2**	0	0.63	0.64	**2995.2**	0	**2995.2**	0	2995.2
neos-584851	13	**-7**	36.36	9.01	9.93	-6	45.45	**-7**	36.36	-11
neos-595905	17	**36148.2**	37.42	798.59	815.59	-	-	-	-	26304.25
neos-709469	1	**493.16**	0	13.99	15.24	**493.16**	0	**493.16**	0	493.16
neos-717614	14	**12396200**	1.19	738.94	757.83	-	-	-	-	12250247.91
neos-796608	12	**-44696100**	7.45	31.53	32.72	-43971750	8.95	**-44696100**	7.45	-48296500
neos-880324	1	**108.67**	0	17.29	18.62	112	3.06	112	3.06	108.67
neos-881765	2	**0**	0	1.32	1.34	**0**	0	**0**	0	0
neos-905856	9	**-6**	0	31.59	32.62	-4	0	**-6**	0	-6
neos-942886	1	**0**	0	45.16	46.12	**0**	0	**0**	0	0
neos-1121679	5	**98**	716.67	5.11	5.15	230	1816.67	791	6491.67	12*
neos-1211578	6	**-77**	0	25.77	26.48	-72	6.49	-75	2.6	-77
neos-1228986	7	**-123**	0	27.06	28.37	-118	4.07	-116	5.69	-123
neos-1337489	7	**-77**	0	26.63	27.16	-72	6.49	-75	2.6	-77
neos-1420205	6	**40**	0	11.91	12.53	53	32.5	343	757.5	40
neos-1425699	10	**3179698977**	0	18.72	19.75	4756138335	49.58	**3179698977**	0	3179698977
neos-1430701	12	**-76**	1.3	27.93	28.61	-69	10.39	-75	2.6	-77
neos-1440447	18	**-99**	1	112.17	113.51	-78	22	-96	4	-100
neos-1460246	2	**2808**	7.75	104.35	106.57	2960	13.58	2860	9.75	2606*
prod1	17	-45	19.64	31.45	32.91	-35	37.5	**-52**	7.14	-56
prod2	6	**-53**	14.52	181.24	183.14	-51	17.74	-43	30.65	-62
rlp1	7	**18**	20	38.51	39.72	21	40	**18**	20	15
roy	5	3242.56	1.05	7.22	7.61	3526.14	9.88	**3208.95**	0	3208.95

*: Best Known Upper Bound
-: Could not find a feasible solution

Table 6.2: Computational Results: ACFM vs Feasibility Pump

Name	Iter	Obj	Name	Iter	Obj
mas74	16	73326	neos-796608	-	-
mas76	1	44877.42	neos-880324	1	108.67
misc07	-	-	neos-881765	2	0
noswot	12	-37	neos-905856	9	-6
pk1	1	28.99	neos-942886	1	0
pp08a	1	9048.56	neos-1121679	12	98
pp08aCUTS	1	8458	neos-1211578	10	-77
rout	12	1111.88	neos-1228986	19	-120
vpm2	9	16	neos-1337489	7	-77
neos2	-	-	neos-1420205	8	40
neos3	15	905.46	neos-1425699	-	-
neos5	1	19	neos-1430701	19	-68
neos-501453	13	52211.89	neos-1440447	-	-
neos-530627	1	2995.2	neos-1460246	2	2808
neos-584851	-	-	prod1	19	-48
neos-595905	-	-	prod2	9	-53
neos-709469	1	493.16	rlp1	-	-
neos-717614	-	-	roy	8	3242.56

-: Could not find a feasible solution in 20 iterations

Table 6.3: Computational Results: ACFM without cuts

	Not Fixing Infeasibility			Fixing Infeasibility			
Name	Iter	Obj	Total time(s)	Iter	Obj	Total time(s)	Time diff(s)
mas74	7	15026.47	13.34	7	15026.47	14.64	1.3
mas76	1	44877.42	3.83	1	44877.42	3.83	0
misc07	13	4795	13.92	13	4795	15.85	1.93
noswot	3	-37	3.76	3	-37	3.79	0.03
pk1	1	28.99	1.13	1	28.99	1.13	0
pp08a	1	9048.56	3.1	1	9048.56	3.1	0
pp08aCUTS	1	8458	4.21	1	8458	4.21	0
rout	4	1111.88	152.93	4	1111.88	158.11	5.18
vpm2	6	15.5	42.65	6	15.5	43.54	0.89
neos5	1	19	0.35	1	19	0.35	0
neos-501474	6	52211.89	17.47	**5**	**52205.74**	18.13	0.66
neos-530627	1	2995.2	0.64	1	2995.2	0.64	0
neos-584851	13	-7	9.93	**11**	-7	9.09	**-0.84**
neos-709469	1	493.16	15.24	1	493.16	15.24	0
neos-796608	12	-44696100	32.72	**11**	-44696100	34.05	1.33
neos-880324	1	108.67	18.62	1	108.67	18.62	0
neos-881765	2	0	1.34	2	0	1.35	0.01
neos-905856	9	-6	32.62	9	-6	33.59	0.97
neos-942886	1	0	46.12	1	0	46.12	0
neos-1121679	5	98	5.15	5	98	6.23	1.08
neos-1211578	6	-77	26.48	6	-77	28.04	1.56
neos-1228986	7	-123	28.37	7	-123	29.93	1.56
neos-1337489	7	-77	27.16	7	-77	29.21	2.05
neos-1420205	6	40	12.53	6	40	14.15	1.62
neos-1425699	10	3179698977	19.75	**9**	3179698977	21.14	1.39
neos-1430701	12	-76	28.61	**10**	-76	29.14	0.53
neos-1440447	18	-99	113.51	**16**	-99	113.23	**-0.28**
neos-1460246	2	2808	106.57	2	2808	106.57	0
prod1	17	-45	32.91	**16**	-52	34.07	1.16
prod2	6	-53	183.14	6	-53	185.99	2.85
rlp1	7	18	39.72	7	18	41.01	1.29
roy	5	3242.56	7.61	5	3242.56	8.03	0.42
						avg	0.83

Table 6.4: Computational Results: ACFM, Fixing infeasibility vs Not fixing infeasibility

Chapter 7

Generalized Branching based on Analytic Centers

Branch-and-Bound [47] is the main general methodology for solving integer programming (IP) problems. Over the years, this methodology has been refined and implemented in commercial solvers. One crucial component of the Branch-and-Bound method is the selection of the branching disjunction. In general commercial solvers, the branching disjunction is based on selecting a single variable among the integer variables that has a fractional value in the current optimal solution of the linear programming relaxation. As shown in [3] and [50], selecting the "best" variable for branching is a major factor in speeding-up the branch-and-bound algorithm. Particularly, selecting the variable that leads to a maximum increase in the lower bound of the subproblem, an approach that is known as strong branching, yields good results in practice.

To provide a detailed description, let us introduce the generic integer problem:

$$\begin{aligned} \min\ & b^T y \\ \text{s.t. } & A^T y + s = c, \\ & s,\ y \geq 0 \\ & y \in \mathbb{Z}^n. \end{aligned} \tag{7.1}$$

that is of interest in this work. Starting from the linear programming relaxation of (7.1) which is obtained by dropping the integrality requirements, the branch-and-bound algorithm proceeds by iteratively creating new problems by branching on variables whose current optimal solution violates the integer requirements. A variable y_j whose value is factional in the current optimal solution is chosen and the feasible region is split into two subproblems corresponding to the addition of the traditional branching constraints $y_j \leq \lfloor y_j \rfloor$ and $y_j \leq \lceil y_j \rceil$ respectively. Such branching constraints are a special case of the more general approach where the constraints $\pi^T y \leq r$ and $\pi^T y \geq r + 1$ are used. The importance of general branching disjunctions was first discussed in [48] and [1] which developed a polynomial time algorithm for general integer programs with a fixed number of variables. Lenstra's algorithm [48] uses the shape of the polyhedron to derive branching constraints in oblique directions and it has been implemented in [15]. Besides using general branching constraints, another major difference is that Lenstra's algorithm generates several branches at a node of the branch-and-bound tree whereas only two branches are created in a classical branch-and-bound. A study of the complexity of Lenstra's algorithm and the shape of the polyhedron has been presented in [74]. In the context of classical branching, Derpich and Vera [23] also used the shape of the polyhedron to set priorities for branching and show that it leads to a significant reduction in the number of nodes that need to be explored. Although using the shape of the polyhedron has been shown to have theoretical and practical importance, methods for deriving generalized branching disjunctions focused

on estimating the change in the objective function value rather than using the shape information. Particularly, Owen and Mehrotra [62] developed a heuristic to derive general disjunctions where an approach similar to strong branching is used to set the variables' coefficients in the disjunctions to $\{-1, 0, 1\}$. Although the proposed approach does not require the basis reduction method as Lenstra's algorithm and was shown to significantly reduce the number of branching, the main disadvantage is the need to solve two linear problems in order to fix the coefficient of each variable. In [53], the problem of finding the general disjunction that maximizes the bound improvement is modeled as a mixed integer program and uses a generic software for its solution. Computational experiments show that the approach of [53] significantly reduce the number of branching nodes however the mixed integer program that is used to generate the disjunctions is $\mathcal{NP}$-hard and computationally expensive to solve in practice [54]. Using the fact that the cutting planes that are used to generate valid inequalities can be used to generate branching disjunctions, Karamanov and Cornuéjols [45] proposes an approach that uses Gomory Mixed Integer (GMI) valid inequalities as general disjunctions in branch-and-bound, while Cornuéjols et al. [16] uses reduce-and-split inequalities to improve the quality of the GMI disjunctions.

In this chapter we present a new algorithm to generate branching disjunctions based on the shape of the polyhedron. At each node, the polyhedron is first approximated using Dikin's ellipsoid which is centered at the analytic centre. We show that finding the minimum-width disjunction over the ellipsoid is equivalent to solving a quadratic problem. As solving a quadratic problem at each node of the branch-and-bound tree is impractical, we use a local search heuristic for its solution. Calculating the analytic center is computationally expensive, and hence we propose to use this method in conjunction with algorithms where the analytic center can be used in other context such as deriving pareto-optimal cuts in Benders Decomposition (Chapter 5). In such a case, the analytic center is calculated once and then used to generate Benders Cuts and to derive branching disjunctions. Furthermore,

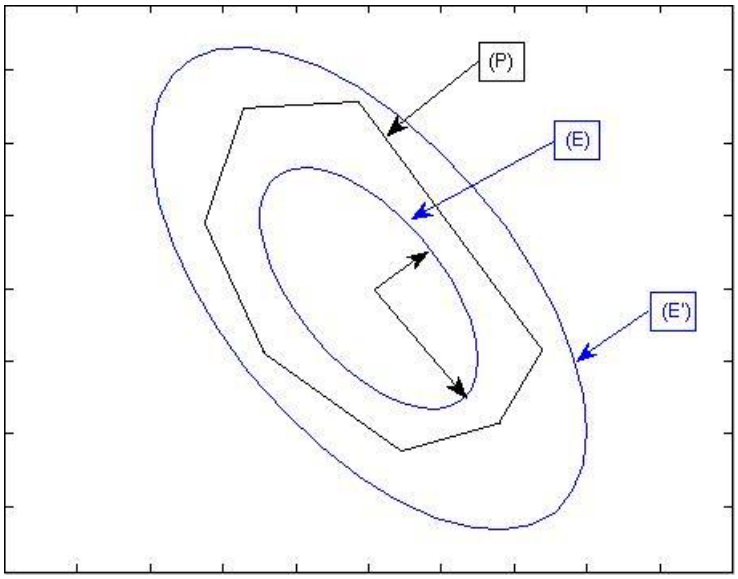

Figure 7.1: Ellipsoidal Approximation

the proposed algorithm can be viewed as an extension of the approach of [23] to general disjunctions.

7.1 Selecting Branching Disjunctions

The efficiency of the branch-and-bound algorithm is typically measured in the number of subproblems generated and hence branching disjunctions are selected so as to minimize this number. However in practice, finding the number of subproblems resulting from taking a particular branching disjunction is extremely hard. While the majority of the research has focused on considering the bound improvement as an indicator on the quality of branching, in this chapter we introduce a new approach based on approximating the shape of the polyhedron using Dikin's ellipsoid and then finding the disjunction that minimizes the width over the ellipsoid.

7.1.1 Ellipsoidal Approximation

To approximate the polyhedron, we use an inscribed ellipse similar to the approach of [23] which showed evidence on the correlation between the size of the axis of the ellipsoid and the width of the polyhedron (Figure 7.1). We denote by P_i the polyhedron of the problem corresponding to node i of the branch-and-bound. For convenience, we drop the subscript i and denote

$$P = \left\{ \ (y, s) : \ \ A^T y + s = c, \ s, \ y \geq 0 \ \right\}.$$

and assume that $A^T y + s = c$ explicitly contain the branching constraints. Given the analytic center $(\overline{y}, \overline{s})$ of P, the associated ellipsoids are $E = \{s : \|\overline{S}^{-1}(s - \overline{s})\| < 1\}$ and $E = \{s : \|\overline{S}^{-1}(s - \overline{s})\| < \gamma\}$ such that $E \subset P \subset E'$ [76]. The analytic center is obtained by solving the following convex problem

$$\begin{aligned} \max \ & \sum_{i=0}^{m} \ln \ s_i \\ \text{s.t.} \ & A^T y + s = c, \\ & s > 0. \end{aligned} \tag{7.2}$$

Problem (7.2) can be solved using Newton method which has been presented in details in Section 2.4.1.

7.1.2 Ellipsoidal Width

An ellipsoid centered at point x is defined by the following equation

$$\xi(x, E) = \{y : y = x + Qs, \ \|s\| \leq 1\}$$

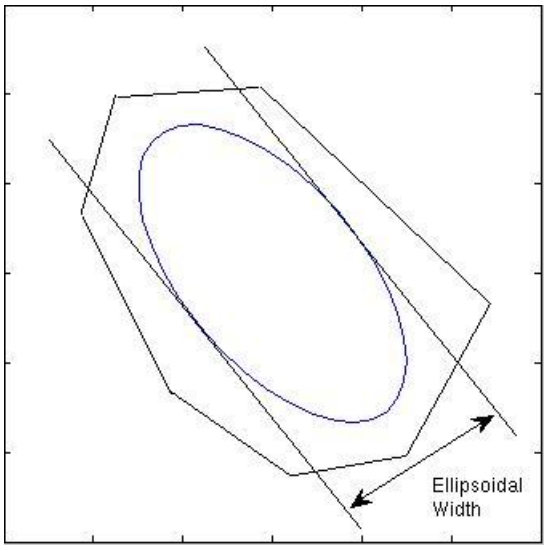

Figure 7.2: Ellipsoidal Width

where Q is a positive definite matrix. The eigenvectors of Q are the principal axis of the ellipsoid and the eigenvalues denote the corresponding norm. The width of the ellipsoid is then equal to the length of the shortest axis, denoted by 2λ, where λ is the smallest eigenvalue of Q [28]. The shortest axis v denotes the thin direction of the polytope as approximated by the ellipsoid and a branching disjunction of the form

$$\pi^T(v)x \leq r \vee \pi^T(v)x \geq r + 1 \tag{7.3}$$

can be obtained by applying a rounding procedure on the elements of v to obtain an integer vector $\pi(v)$ that can be used for branching. However, we note that a rounding procedure will typically not guarantee that the resulting integer vector $\pi(v)$ is a thin direction. Furthermore, finding the eigenvectors and eigenvalues is computationally expensive. Alternatively as shown in Figure 7.1.1, we define the ellipsoidal width $w(P, v)$ of P in a

direction v as the width of the ellipsoidal approximation of P in the direction v

$$w(P,v) = \max\{v^T y : y \in E\} - \min\{v^T y : y \in E\} \tag{7.4}$$

whose solution is given by the following theorem.

Theorem 2.

$$w(P,v) = \max_y\{v^T y : y \in E\} - \min_y\{v^T y : y \in E\} = 2\sqrt{v^T(AS^{-2}A^T)^{-1}v} \tag{7.5}$$

Proof: Let us write $\max_y\{v^T y : y \in E\}$ as $\max_y\{v^T y : (y-\overline{y})^T(AS^{-2}A^T)^{-1}(y-\overline{y}) < 1\}$, whose Lagrangian is given by

$$\max_{y,\lambda>0} v^T y + \lambda((y-\overline{y})^T(AS^{-2}A^T)^{-1}(y-\overline{y}) - 1). \tag{7.6}$$

The derivatives with respect to y and λ are given by

$$v + 2\lambda(AS^{-2}A^T)^{-1}(y-\overline{y}) = 0 \tag{7.7}$$

$$(y-\overline{y})(AS^{-2}A^T)^{-1}(y-\overline{y}) = 1 \tag{7.8}$$

whose solution is

$$(2\lambda)^2 = v^T(AS^{-2}A^T)^{-1}v. \tag{7.9}$$

Taking the positive root for λ, leads to

$$y_{max} = \overline{y} + \sqrt{\frac{1}{v^T(AS^{-2}A^T)^{-1}v}}(AS^{-2}A^T)^{-1}v. \tag{7.10}$$

The maximum objective $v^T y_{max}$ is then

$$a^T \overline{y} + \sqrt{v^T (AS^{-2}A^T)^{-1} v}. \tag{7.11}$$

To find the minimum objective one should keep the negative root as λ to get

$$a^T \overline{y} - \sqrt{v^T (AS^{-2}A^T)^{-1} v}. \tag{7.12}$$

and

$$\max\{v^T y : y \in E\} - \min\{v^T y : y \in E\} = 2\sqrt{v^T (AS^{-2}A^T)^{-1} v} \tag{7.13}$$

□

Finding the vector v that minimizes the ellipsoidal width is equivalent to solving the convex quadratic problem

$$\min_v \; v^T (AS^{-2}A^T)^{-1} v \tag{7.14}$$

$$\text{s.t. } v \neq 0. \tag{7.15}$$

The condition $v \neq 0$ can be replaced by $\sum_i v_i \geq 1$. In this chapter, we consider disjunctions of the form

$$\pi^T x \leq r \vee \pi^T x \geq r + 1 \tag{7.16}$$

where $\pi \in \{-1, 0, 1\}$ and $r \in \mathbb{Z}$ which are valid for any mixed integer program. Hence, the disjunction that minimizes the ellipsoidal width is the solution of

$$\min_{\pi} \ \pi^T (AS^{-2}A^T)^{-1}\pi \tag{7.17}$$

$$\text{s.t.} \ \sum_i \pi_i \geq 1 \tag{7.18}$$

$$\pi \in \{-1, 0, 1\}. \tag{7.19}$$

Solving problem (7.17)-(7.19) to optimality at each node of the branch-and-bound tree is computationally intractable. For that, we use a local search heuristic to find good disjunctions.

7.1.3 A Local Search Heuristic

In this section we introduce a local search algorithm that finds branching disjunctions $\pi^T x \leq r \vee \pi^T x \geq r + 1$ that minimize the the ellipsoidal width $\pi^T (AS^{-2}A^T)^{-1}\pi$. The algorithm starts by sorting the variables according to a decreasing order of the ellipsoidal width corresponding to the single-variable disjunction. Note that this is equivalent to sorting the variables according to the corresponding values in the diagonal of the square matrix $(AS^{-2}A^T)^{-1}$. The algorithm first selects the minimum value single-variable $\overline{\pi}$ disjunction and then attempts to find a better disjunction by flipping each element $\overline{\pi}_i$ between $+1$, -1, and 0. A similar algorithm was used in [62] to find general disjunctions that maximize the lower bound. A major difference however is that the approach of [62] requires the solution of two linear programming relaxations when each element is flipped in contrast to the proposed approach which only requires the evaluation of the ellipsoidal width $\pi^T (AS^{-2}A^T)^{-1}\pi$. The local search heuristic is outlined next.

1. Initialization:
   1.1 Calculate $Q = (AS^{-2}A^T)^{-1}$
   1.2 Let N denote the set of integer variables whose value in the current optimal solution is non integer, i.e. $N = \{i|\overline{x}_i \notin \mathbb{Z}\}$.
   1.2 Sort the variables in N in ascending order of the corresponding value in the diagonal of the matrix Q. Let n(j) denote the variable in position j of the sorted array.
   1.3 Initialize the disjunction as $\pi = e_{n(1)}$ where e_i is a vector of all zeros with entry i set to 1.
2. For $j = 2;\ j \leq |N|;\ j = j + 1$
   2.1 Let $\overline{\pi} = \pi$ and set $\overline{\pi}_{n(j)} = 1$
   2.2 If $\overline{\pi}^T(AS^{-2}A^T)^{-1}\overline{\pi} \leq \pi^T(AS^{-2}A^T)^{-1}\pi$
   Set $\pi = \overline{\pi}$
   2.3 Let $\overline{\pi} = \pi$ and set $\overline{\pi}_{n(j)} = -1$
   2.4 If $\overline{\pi}^T(AS^{-2}A^T)^{-1}\overline{\pi} \leq \pi^T(AS^{-2}A^T)^{-1}\pi$
   Set $\pi = \overline{\pi}$
3. Calculate $r = \lfloor \pi\overline{x} \rfloor$ and return the disjunctions $\pi^T y \leq r \ \vee \ \pi^T y \geq r + 1$

7.2 Generalized Branching in Benders-Branch-and-Cut

Although generalized branching has been shown to significantly reduce the size of branch-and-bound trees, the majority of the approaches discussed in the literature suffer from large computational burden limiting the use of generalized branching in practice. In the presented approach, the computational bottleneck lies in the calculation of the analytic center which motivates the use of our proposed generalized branching scheme within other approaches where the analytic center is available. The Benders-branch-and-cut method introduced in Chapter 5 uses the analytic center to derive strong Benders inequalities within a branch-and-cut tree. Particularly, the Benders-branch-and-cut method solves the

problem

$$\min\ d^T y + \theta \tag{7.20}$$

$$\text{s.t. } A_1^T y \leq c_1, \tag{7.21}$$

$$(c_2 - M^T y)\lambda \leq \theta \quad \lambda \in H^P, \tag{7.22}$$

$$(c_2 - M^T y)\mu \leq 0 \quad \mu \in H^R, \tag{7.23}$$

$$y \geq 0 \text{ and integer.} \tag{7.24}$$

where cuts (7.22) and (7.23) are generated inside a branch-and-cut tree using the analytic center of the relaxation at each node of the tree. Since the analytic center has been calculated and hence no additional computational effort is required, we propose to use it to derive generalized branching constraints rather than enforcing the integrality constraints using classical branching. The modified Benders-branch-and-cut method follows

1. Initialization: Set the incumbent $Z_U = \infty$.
2. Select a non fathomed node from the tree. If non exists, stop.
   - 2.1 Initialize $z_l = -\infty$ and $z_u = Z_U$.
   - 2.2 While $|z_u - z_l| > \varepsilon$, $z_l < Z_U$, and the relaxed master problem is feasible
     - 2.1 Find the analytic center y^{ac} and the corresponding dual x^{ac}.
     - 2.2 Using y^{ac}, solve the subproblem to generate a Benders cut.
     - 2.3 Add the generated Benders cut to the relaxed master problem.
     - 2.4 If y^{ac} doesn't violate the generated cut, update the upper bound $z_u = d^T y^{ac}$.
     - 2.4 Update the lower bound $z_l = \max\{z_l, c^T x^{ac}\}$.
   - 2.2 End While.
   - 2.2 If $z_l > Z_U$ or if the relaxed master problem is infeasible, fathom the node.
     - 2.2.1 Go back to Step 2.
   - 2.2 If y^{ac} satisfies the integrality constraints, fathom the node.
     - 2.2.1 Update $Z_U = \min\{Z_U, d^T y^{ac}\}$, and go back to Step 2.
3. Use y^{ac} to find a generalized disjunction.

4. Create two nodes by branching using the derived disjunctions and go back to Step 2.

In the following section we evaluate the computational performance of the proposed branching approach in Benders-branch-and-cut and in solving general mixed integer problems.

7.3 Computational Results

In this section, we present computational results conducted on a Sunblade 2500 workstation with a 1.6 GHz processor and 2 Gb of RAM. We first present computational results on the integration of the proposed generalized branching approach in Benders-branch-and-cut. We then provide computational results on 54 problems collected from MIPLIB [4] and CORAL [49].

7.3.1 Generalized Branching in Benders-Branch-and-Cut

As discussed in Section 7.2, the proposed branching approach is most computationally efficient with branch-and-cut methods where the analytic center is available hence saving the additional computational burden when deriving the proposed general branching disjunctions. In this section, we evaluate the performance of ACCPM-Benders-Branch-and-Cut with the proposed generalized branching approach and with classical branching by solving 30 instances of the MCFND problem. The instances are generated using the mulgen generator [19]. The computational results are reported in Table 7.1. The name of each problem indicates the size of the considered network. For instance, $p\_n\_m\_k$ denotes a network with n nodes, m arcs, and k commodities. Networks with 10 and 15 nodes are considered while the number of arcs is varied between 30, 40, 50, 60, and 70. Additionally, the number of commodities is varied between 2, 3 and 4.

	General Branching: ACCPM-Benders-b&c (GABBC)					Classic Branching: ACCPM-Benders-b&c (CABBC)					
Name	Nodes	Cuts	Cuts0	Cuts AVG	CPU	Nodes	Cuts	Cuts0	Cuts AVG	CPU	Sol
p_10_30_2	19	72	23	2.58	0.39	19	75	23	2.89	0.47	18721
p_10_40_2	23	115	35	3.48	1.19	23	145	35	6.59	1.59	16983
p_10_50_2	27	135	29	3.93	4.04	33	164	29	5.12	6.05	9688
p_10_60_2	23	86	33	2.30	3.68	23	119	33	5.41	5.92	21902
p_10_70_2	17	89	27	3.65	5.29	17	95	27	5.94	6.12	15958
p_10_30_3	51	241	35	4.04	5.49	63	338	35	5.45	8.38	22281
p_10_40_3	51	147	37	2.16	4.40	53	356	37	6.85	12.07	20220
p_10_50_3	39	410	43	9.41	21.64	45	511	43	11.61	29.53	13970
p_10_60_3	29	152	39	3.90	10.90	29	172	39	6.14	13.37	24129
p_10_70_3	53	490	41	8.47	63.22	73	935	41	12.99	129.99	21690
p_10_30_4	45	508	50	10.18	16.36	61	552	50	9.2	21.09	29177
p_10_40_4	85	1427	50	16.20	359.92	233	2729	50	11.76	742.89	27995
p_10_50_4	59	1785	50	29.41	366.84	121	2016	50	16.8	454.63	31219
p_10_60_4	55	3183	41	57.13	2262.42	329	5210	41	15.88	4067.88	29048
p_10_70_4	53	1615	33	29.85	649.67	91	2567	33	28.52	1135.31	25086
p_15_30_2	25	32	27	0.20	0.17	25	57	27	1.25	0.32	41599
p_15_40_2	19	25	24	0.05	0.50	19	96	24	5.33	2.32	21257
p_15_50_2	21	96	29	3.19	3.00	21	105	29	5.25	3.75	22350
p_15_60_2	73	687	30	9.00	56.99	101	810	30	8.1	72.05	31532
p_15_70_2	21	403	27	17.90	36.00	21	428	27	21.4	40.37	19204
p_15_30_3	19	158	37	6.37	1.46	33	164	37	5.12	1.58	23723
p_15_40_3	49	641	46	12.14	61.21	71	1265	46	18.07	133.45	32400
p_15_50_3	23	144	22	5.30	4.73	27	152	22	5.85	5.6	36347
p_15_60_3	303	4266	66	13.86	3439.34	471	4945	66	10.52	4135.12	28476
p_15_70_3	75	2277	47	29.73	1883.32	151	4159	47	27.73	3731.07	27557
p_15_30_4	23	206	40	7.22	5.68	45	470	40	10.68	14.85	67389
p_15_40_4	33	1091	53	31.45	94.96	65	1165	53	18.2	106.06	47009
p_15_50_4	31	602	52	17.74	76.73	67	1413	52	21.41	238.03	54074
p_15_60_4	39	1309	48	32.33	248.24	51	1405	48	28.1	277.69	43115
p_15_70_4	33	531	48	14.64	48.79	33	612	48	19.12	59.91	32781
Avg (ratio)	0.76	0.74	-	0.94	0.67						

Table 7.1: MCFND Computational Results

Comparing GABBC to CABBC, we observe that GABBC explores less nodes than CABBC for the majority of the tested instances. On average, GABBC explores 24% less nodes than CABBC. Even for the problems where GABBC and CABBC explores the same number of nodes, GABBC generates fewer cuts than GABBC leading to a reduction in the total computational time. On average GABBC requires 26% less cuts than CABBC and requires 33% less computational time. We observe that for hard instances such as $p\_10\_60\_4$, $p\_15\_60\_3$, and $p\_15\_70\_3$, GABBC leads to a significant reduction of upto 50% in the total computational time.

7.3.2 Generalized Branching in general mixed integer programming

In this section, we evaluate the performance of our proposed branching scheme in solving general mixed integer programs. We conduct two sets of testing, the first using a pure branch-and-bound algorithm where no additional cutting planes or heuristics are used. The second is based on implementing our proposed branching scheme in the CPLEX branch-and-cut algorithm with all the default cutting planes, heuristics, and settings enabled.

Experiments using a pure branch-and-bound

To evaluate the performance, we present computational results using our proposed braching rule in a pure branch-and-bound. The branch-and-bound algorithm is implemented in C and uses a depth first search strategy. We compare using our branching scheme against branching according to the most fractional variable. Similar to [23], we conduct our testing on randomly generated instances and on multidimensional knapsack problems from OR-Lib [10]. The randomly generated instances are denoted by t_n. The values for the matrix A are distributed uniformly between zero and one. The vector c is generated as to guarantee feasibility of the polyhedron, plus a random perturbation. The b vector

				General Branching		Strong Branching	
Name	n	m	p	Nodes	CPU	Nodes	CPU
t1	30	15	50	28.72	0.17	34.92	0.19
t2	60	30	50	99.27	0.92	119.16	1.02
t3	80	40	50	145.14	1.66	175.4	1.85
t4	100	50	50	297.77	5.18	340.52	5.47
t5	120	60	50	453.91	9.24	528.6	9.93
t6	140	70	50	442.69	10.62	496.04	10.98
t7	160	80	50	490.72	18.67	580.08	20.37
t8	180	90	50	862.64	34.97	1039.96	38.92
t9	200	100	50	1113.26	52.47	1296.52	56.41
Avg (ratio)				0.85	0.92		
Mknap1_1	10	6	1	16	0.06	23	0.07
Mknap1_2	10	10	1	20	0.09	25	0.1
Mknap1_3	10	15	1	121	0.54	159	0.62
Mknap1_4	10	20	1	147	0.65	217	0.83
Mknap1_5	10	28	1	373	1.69	527	2.07
Mknap1_6	5	39	1	346	2.07	509	2.64
Mknap1_7	5	50	1	372	2.55	515	3.06
Avg (ratio)				0.72	0.83		
Mknapcb1_1	5	100	1	123370	1141.75	190495	1656.12
Mknapcb1_2	5	100	1	216111	1998.25	330139	2867.6
Mknapcb1_3	5	100	1	30674	247.04	47107	356.39
Mknapcb1_4	5	100	1	212619	2024.13	379203	3391.22
Mknapcb1_5	5	100	1	132471	1248.56	195289	1729.08
Mknapcb1_6	5	100	1	228364	2137.69	298651	2626.2
Mknapcb1_7	5	100	1	89511	800.43	150175	1261.52
Mknapcb1_8	5	100	1	197288	1906.33	273961	2486.76
Mknapcb1_9	5	100	1	151621	1421.67	232261	2045.8
Mknapcb1_10	5	100	1	192566	1793.53	308165	2696.26
Mknapcb1_11	5	100	1	99437	903.73	172483	1472.6
Mknapcb1_12	5	100	1	328072	2638.42	566153	4277.17
Mknapcb1_13	5	100	1	182053	1395.5	266791	1921.11
Mknapcb1_14	5	100	1	117444	926.87	181037	1342.15
Mknapcb1_15	5	100	1	110374	836.17	187823	1336.68
Mknapcb1_16	5	100	1	98441	750.68	133267	954.66
Mknapcb1_17	5	100	1	280500	2267.52	383459	2911.95
Mknapcb1_18	5	100	1	371874	2982.67	566589	4269
Mknapcb1_19	5	100	1	147316	1110.56	218303	1545.96
Mknapcb1_20	5	100	1	74368	543.76	118373	813.06
Mknapcb1_21	5	100	1	83467	613.28	130941	903.8
Mknapcb1_22	5	100	1	221645	1696.19	280671	2017.73
Mknapcb1_23	5	100	1	135944	1097.2	193659	1468.29
Mknapcb1_24	5	100	1	75153	580.68	95849	695.71
Mknapcb1_25	5	100	1	49565	386.35	75693	554.25
Mknapcb1_26	5	100	1	210505	1633.89	309151	2254.14
Mknapcb1_27	5	100	1	82952	581.28	113955	750.14
Mknapcb1_28	5	100	1	119490	910.72	169159	1211.14
Mknapcb1_29	5	100	1	92651	696.65	118477	836.84
Avg (ratio)				0.67	0.71		

Table 7.2: Computational Results: Pure Branch-and-Bound

was generated randomly with normal distribution with median zero and variance one. For the multidimensional knapsack problems, we conduct our testing on the mknap1 and the mknap2 instances that are available on OR-Lib [10]. The Following results are reported in Table 7.2:

$Name$: Instance Name.
n : Number of rows.
m : Number of columns.
p : Number of problems solved.
$Nodes$: Number of nodes of the search tree.
CPU : Total computational time in seconds.

For the randomly generated instances, we generate 50 problems of each size and report the average results. For the 45 tested instances, the proposed general branching approach outperforms strong branching. For the randomly generated instances, the proposed branching approach explored 15% less branching nodes than strong branching and consumed 8% less computational time. For the Mknap1 instances, the proposed branching approach explored 28% less branching nodes than strong branching and consumed 17% less computational time. The difference is more significant for the harder Mknapcb1 instances where the proposed branching branching approach explored 33% less nodes and consumed 29% less computational time. In a pure branch-and-bound, the proposed general branching approach outperforms strong branching both in terms of number of nodes explored and in computational time. However, it is known that the branching strategies are interrelated with cutting planes and heuristics and hence the overall performance. In the following section, we conduct computational experiments on general mixed integer problems by implementing our proposed branching method in a state of the art branch-and-cut algorithm.

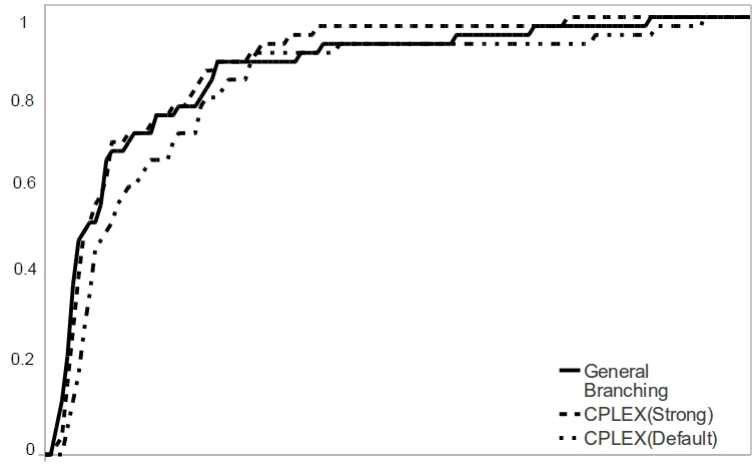

Figure 7.3: Performance profile for number of nodes solved in the branch-and-bound

Experiments using a commercial branch-and-cut

This section presents computational results performed on 54 problems collected from MIPLIB [4] and CORAL [49]. The instances with the corresponding number of binary variables (B), the number of integer variables (I), the total number of variables (n), and the total number of constraints (m) are described in Table 7.3. The proposed branching scheme implemented in C using the callback functions of CPLEX 11. Since branching strategies can affect the performance of several other methods such as cutting plane generation, heuristics, and node selection which are employed in the solutions of MIP, we used the default CPLEX parameters in our testing to evaluate the global effect of our proposed branching approach.

In contrast to the pure branch-and-bound results where the proposed branching approach outperformed strong branching, the results displayed in Table 7.4 show that the cutting planes and the heuristics affect the performance of the branching method. Figure 7.3 shows the performance profile for the data in Table 7.4. For the 54 tested instances, our proposed branching approach explored less branching in 26 instances while strong branching out-

performed in 25 instances. For 3 instances, the proposed branching approach explored the same number of nodes as strong branching while in 7 instances cplex default branching outperformed both the proposed branching approach and strong branching. On average, the proposed branching approach explored 4% less nodes than strong branching and 46% less nodes than cplex default branching. The computational time of our proposed branching approach is however more significant than CPLEX. This mainly due to our implementation of the analytic center calculation method. Our testing is only aimed at evaluating the viability of the approach and the stability of the method, however to outperform state of the art commercial solvers, an efficient implementation of the analytic center method will be necessary. We note that strong branching is explored 44% less nodes that cplex default branching but was 88% more computationally expensive.

Evaluating the local search heuristic

In this section, we evaluate the effect of solving problem (7.17)-(7.19) to optimality instead of using the local search heuristic. Since solving problem (7.17)-(7.19) takes a significant amount of computational time, we conduct testing on the problems that required less than 500 nodes to be explored when the local search heuristic was used. The results are displayed in Table 7.5. We notice that for the majority of the tested problems, solving (7.17)-(7.19) or using the local search heuristic lead to the same number of nodes. In only 11 of the 38 problems, local search resulted in more nodes while for 4 problems local search resulted to less nodes. On average using local search lead to 6% increase in the number of nodes. The difference is however more significant in the computational time where solving problem (7.17)-(7.19) leads to 78% increase in the computational time on average.

7.4 Conclusion

We presented a new approach to derive general branching disjunctions based on the shape of the polyhedron. We use the Dikin's ellipsoid which we calculate using the analytic center to find a disjunction that minimizes the width of the polyhedron. We formulate a quadratic program to find the minimum width disjunction and proposed a local search heuristic for its solution. Calculating the analytic center is computationally expensive, and hence the proposed branching approach is most computationally efficient with branch-and-cut methods where the analytic center is available such as the ACCPM-Benders-Branch-and-Cut method. We conduct computational testing on general mixed integer problems using a pure branch-and-bound algorithm and using CPLEX branch-and-cut with all default features enabled. In a pure branch-and-bound method, our proposed approach outperforms strong branching. Using the CPLEX branch-and-cut implementation, our proposed approach outperformed strong branching in 26 out of the 54 tested instances and outperformed cplex default branching in 44 instances at the expense of taking longer computational time.

Exploring a hybrid branching rule that takes into account the shape of the polyhedron as well as other information such as the objective function coefficients and effect on the resulting relaxation bounds as in strong branching is a promising future research direction.

Name	n	m	B	I	Name	n	m	B	I
10teams	230	2025	1800	0	modglob	291	422	324	98
aflow30a	479	842	421	0	neos-1211578	356	260	130	0
bell3a	123	133	39	32	neos-1228986	356	260	130	0
bell3b	123	133	39	32	neos-1337489	356	260	130	0
bell4	105	117	34	30	neos-530627	113	103	0	28
bell5	91	104	30	28	neos-584851	661	445	405	0
blend2	274	353	231	33	neos-880324	348	261	232	0
bm23	20	27	27	0	neos6	1036	8786	8340	0
dcmulti	290	548	75	0	nw04	36	87482	87482	0
fiber	363	1298	1254	0	p0033	16	33	33	0
fixnet4	478	878	378	0	p0201	133	201	201	0
fixnet6	478	878	378	0	p0282	241	282	282	0
flugpl	18	18	0	11	p0291	252	291	291	0
gesa2	1392	1224	240	168	p2756	755	2756	2756	0
gesa3	1368	1152	216	168	pipex	25	48	48	0
gesa3_o	1224	1152	336	336	pp08aCUTS	246	240	64	0
l152lav	97	1989	1989	0	qnet1	503	1541	1288	129
lseu	28	89	89	0	ran12x21	285	504	252	0
misc01	54	83	82	0	ran13x13	195	338	169	0
misc02	39	59	58	0	rgn	24	180	100	0
misc03	96	160	159	0	roy	162	149	50	0
misc05	300	136	74	0	sample2	45	67	21	0
misc07	212	260	259	0	sentoy	30	60	60	0
mod008	6	319	319	0	set1ch	492	712	240	0
mod010	146	2655	2655	0	stein15	36	15	15	0
mod011	4480	10958	10862	96	stein27	118	27	27	0
mod013	62	96	48	0	vpm2	234	378	168	0

Table 7.3: Test Problems

	General Branching		CPLEX (Strong)		CPLEX (Default)	
Name	Nodes	CPU(s)	Nodes	CPU(s)	Nodes	CPU(s)
10teams	960	21044.3	854	61.54	**140**	3.59
aflow30a	2513	6261.38	**1415**	34.62	2885	14.93
bell3a	**20165**	399.47	24567	1.73	25668	1.93
bell3b	2139	140.18	1593	0.35	**1516**	0.32
bell4	**411**	22.36	550	0.35	818	0.22
bell5	**622**	28.03	793	0.13	854	0.13
blend2	897	6.32	**746**	2.25	1071	1.41
bm23	43	0.78	56	0.3	**34**	0.08
dcmulti	**39**	18.04	49	0.51	65	0.31
fiber	**26**	2.25	48	0.23	60	0.22
fixnet4	**7**	1.08	17	0.25	31	0.28
fixnet6	**24**	12.8	35	0.69	37	0.73
flugpl	127	8.27	112	0.01	**88**	0.01
gesa2	**41**	19.8	61	0.45	80	0.29
gesa3	45	29.93	**29**	0.35	37	0.73
gesa3_o	**30**	19.96	31	0.42	62	0.52
l152lav	**113**	114.96	158	1.99	624	1.65
lseu	77	4.33	**54**	0.08	100	0.04
misc01	124	6.83	**90**	0.57	186	0.20
misc02	**54**	0.21	378	0.28	378	0.26
misc03	133	10.7	**81**	3.45	217	0.59
misc05	**28**	1.45	41	0.17	93	0.08
misc07	5587	713.03	**2415**	7.91	9819	9.87
mod008	256	8.5	**163**	0.16	316	0.08
mod010	**11**	2.2	18	0.85	25	0.73
mod011	**35**	10.55	82	33.24	74	25.23
mod013	**13**	0.79	45	0.03	101	0.02
modglob	126	31.25	**63**	0.16	164	0.12
neos-1211578	**26569**	3353.73	26570	62.2	14320	5.74
neos-1228986	**41533**	10184.6	41966	85.26	44080	12.93
neos-1337489	26576	3353.73	26570	62.2	**14320**	5.74
neos-530627	948573	52657.1	**769038**	13.27	781220	13.53
neos-584851	**48**	82.18	52	2.33	229	1.21
neos-880324	**106**	9.42	131	0.57	336	0.22
neos6	**152**	20037	164	40.95	802	56.21
nw04	**28**	314.5	32	29.75	189	29.44
p0033	23	23.08	**20**	0.02	30	0.02
p0201	23	2.82	**19**	1.04	802	0.01
p0282	**17**	1.16	**17**	0.36	54	0.25
p0291	13	0.09	**11**	0.06	21	0.1
p2756	272	367.95	**162**	0.39	548	0.63
pipex	**8**	0.25	**8**	0.13	20	0.12
pp08aCUTS	431	63.64	**403**	2.37	1395	1.67
qnet1	**8**	3.24	25	1.22	51	1.14
ran12x21	11969	2652.48	**8949**	163.53	30459	54.80
ran13x13	7751	1344.1	**1958**	26.5	12153	15.50
rgn	819	34.2	681	0.41	**565**	0.13
roy	**30**	4.01	38	0.05	47	0.12
sample2	**18**	0.31	44	0.03	76	0.01
sentoy	**31**	0.64	36	0.12	89	0.09
set1ch	142	34.4	137	1.09	**89**	0.01
stein15	36	0.36	**30**	0.07	57	0.05
stein27	996	20.67	**231**	0.37	1442	0.39
vpm2	**992**	91.96	**992**	1.79	1153	0.47
Avg (ratio, Strong)	0.96	29.8				
Avg (ratio, Default)	0.54	55.88	0.56	1.88		

Table 7.4: Computational Results: CPLEX Branch-and-Cut

	General Branching (local Search)		General Branching (Solve (7.17)-(7.19))	
Name	Nodes	CPU(s)	Nodes	CPU(s)
bell4	411	22.36	411	385.09
bm23	43	0.78	43	7.59
dcmulti	39	18.04	39	37.44
fiber	26	2.25	26	5.18
fixnet4	7	1.08	7	2.04
fixnet6	24	12.8	24	14.31
flugpl	127	8.27	127	57.43
gesa2	41	19.8	41	67.08
gesa3	45	29.93	40	56.58
gesa3	30	19.96	30	47.88
l152lav	113	114.96	113	205.16
lseu	77	4.33	77	52.62
misc01	124	6.83	122	143.50
misc02	54	0.21	35	23.94
misc03	133	10.7	133	59.56
misc05	28	1.45	17	7.09
mod008	256	8.5	256	58.74
mod010	11	2.2	17	4.44
mod011	35	10.55	35	20.33
mod013	13	0.79	5	5.22
modglob	126	31.25	170	130.07
neos-584851	48	82.18	48	137.71
neos-880324	106	9.42	106	57.21
neos6	152	20037	152	20170.88
nw04	28	314.5	28	360.98
p0033	23	23.08	22	23.15
p0201	23	2.82	23	4.73
p0282	17	1.16	17	10.12
p0291	13	0.09	13	8.61
p2756	272	367.95	221	542.29
pipex	8	0.25	8	0.76
pp08aCUTS	431	63.64	421	418.18
qnet1	8	3.24	7	9.76
roy	30	4.01	30	5.01
sample2	18	0.31	16	4.92
sentoy	31	0.64	17	13.87
set1ch	142	34.4	159	43.64
stein15	36	0.36	39	36.67
Avg (ratio)	1.06	0.22		

Table 7.5: Computational Results: Evaluating the effect of the local search heuristic

Chapter 8

Conclusion

This research explores the use of interior point concepts in integer programming through the analytic center cutting plane method. We first reviewed the use of ACCPM within branch-and-price and then presented three approaches to use interior point methods in integer programs. The first integrates the analytic center cutting plane method in a branch-and-cut algorithm based on Benders decomposition. The second exploits ACCPM in a new heuristic for integer programming. The third derives good general branching disjunctions using an inscribed ellipsoid centered at the analytic center.

To design a branch-and-cut algorithm based on Benders decomposition, we first show that the Benders cuts that are generated at any node are global cuts. Second, we show that the Benders cuts that are generated using the analytic center of the master problem are pareto-optimal cuts. The computational results confirm the Benders cuts that are generated using ACCPM are tighter than the Benders cuts that are generated using Kelley's cutting plane method. Future work on the Benders-Branch-and-Cut method will focus on using a sophisticated branch-and-cut implementation based on CPLEX callbacks or other open source packages such as CBC from COIN-OR or SCIP. Ultimately, the aim is to evaluate the performance of our method on solving general mixed integer problems.

Motivated by the fact that rounding an interior point is more likely to result in a feasible integer solution compared to rounding an extreme point, we present a heuristic for finding feasible solutions for general mixed integer problems. The analytic center feasibility method is based on searching around two line segments, connecting the analytic center to two extreme points of the LP relaxation of the MIP. Using weights, the analytic center is moved iteratively and rounded to the nearest integer. The analytic center feasibility method found better feasible solutions than the feasibility pump on a number of problems. Extending the method to non-linear MIP is particularly interesting for future work.

Finally, we present a new approach to generate good general branching constraints by using an inscribed ellipsoid to approximate the shape of the polyhedron. We use the disjunction that has a minimum width on the ellipsoid as branching constraints which can be found by solving a quadratic problem. Since solving a quadratic problem at each node of the branch-and-bound tree is computationally impractical, we present a local search heuristic for its solution. Computational testing showed that the proposed general branching approach leads to a significant reduction in the number of nodes that are explored by branch-and-bound. Future research will focus on developing a hybrid branching approach that combines general branching and classical techniques to benefit from the advantages of both methods in the branch-and-bound framework. Additionally, reducing the computational time is essential to make the method effective in practice.

References

[1] K. Aardal and F. Eisenbrand. Integer programming, lattices, and results in fixed dimension. In G. N. K. Aardal and R. Weismantel, editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*, pages 171–243. Elsevier, 2005.

[2] T. Achterberg and T. Berthold. Improving the feasibility pump. *Discrete Optimization*, 4(1):77–86, 2007.

[3] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[4] T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. Technical report, Zuse Institute Berlin, 2005. (http://miplib.zib.de/).

[5] O. Bahn, J.-L. Goffin, J.-P. Vial, and O. du Merle. Experimental behavior of an interior point cutting plane algorithm for convex programming: An application to geometric programming. *Discrete Applied Mathematics*, 49:3–23, 1994.

[6] E. Balas and C. Martin. Pivot-and-complement: A heuristic for 0-1 programming. *Management Science*, 26:86–96, 1980.

[7] E. Balas, S. Ceria, M. Dawande, F. Margot, and G. Pataki. OCTANE: A new heuristic for pure 0-1 programs. *Operations Research*, 49(2):207–225, 2001.

[8] E. Balas, S. Schmieta, and C. Wallace. Pivot and shift-a mixed integer programming heuristic. *Discrete Optimization*, 1:3–12, 2004.

[9] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1998.

[10] J. Beasley. OR-library: Distributing test problems by electronic mail. (http://people.brunel.ac.uk/ mastjjb/jeb/info.html).

[11] J. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[12] L. Bertacco, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4(1):63–76, 2007.

[13] B. Borchers and J. E. Mitchell. Using an interior point method in a branch-and-bound algorithm for integer programming. Technical Report Technical Report 195, Rensselaer Polytechnic Institute, Troy, NY 12180, March 1991, Revised July 1992.

[14] O. Briant, C. Lemaréchal, P. Meurdosoif, S. Michel, N. Perrot, and F. Vanderbeck. Comparison of bundle and classical column generation. *Mathematical Programming*, 113(2):299–344, 2008.

[15] W. Cook, T. Rutherford, H. Scarf, and D. Shallcross. An implementation of the generalized basis reduction algorithm for integer programming. *ORSA Journal on Computing*, 5:206–212, 1993.

[16] G. Cornuéjols, L. Liberti, and G. Nannicini. Improved strategies for branching on general disjunctions. *Mathematical Programming*, pages 1–23, 2009.

[17] A. M. Costa. A survey on Benders decomposition applied to fixed-charge network design problems. *Computers and Operations Research*, 32(6):1429–1450, 2005.

[18] G. Cote and M. Laughton. Large-scale mixed integer programming: Benders-type heuristics. *European Journal of Operational Research*, 16:327–333, 1984.

[19] T. G. Crainic, A. Frangioni, and B. Gendron. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics*, 112(1-3):73–99, 2001.

[20] H. Crowder, E. Johnson, and M. Padberg. Solving large-scale 01 linear programming problems. *Operations Research*, 31:803–834, 1983.

[21] J. Czyzyk, M. P. Mesnier, and J. J. More. The NEOS server. *IEEE Computational Science and Engineering*, 5(3):68–75, 1998.

[22] G. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

[23] I. Derpich and J. R. Vera. Improving the efficiency of the branch and bound algorithm for integer programming based on flatness information. *European Journal of Operational Research*, 174(1):92 – 101, 2006.

[24] J. Desrosiers, F. Soumis, and M. Desrochers. Routing with time windows by column generation. *Networks*, 14:545–565, 1984.

[25] S. Elhedhli and J.-L. Goffin. The integration of an interior-point cutting plane method within a branch-and-price algorithm. *Mathematical Programming*, 100(2):267–294, 2004.

[26] J. Elzinga and T. Moore. A central cutting plane algorithm for the convex programming problem. *Mathematical Programming*, 8:134–145, 1975.

[27] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.

[28] L. Gao and Y. Zhang. Computational experience with lenstra's algorithm. Technical Report TR02-12, Department of Computational and Applied Mathematics, Rice University, 2002.

[29] A. M. Geoffrion and G. Graves. Multicommodity distribution system design by benders decomposition. *Management Science*, 20:822–844, 1974.

[30] F. Glover. Tabu search - Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[31] F. Glover. Tabu search - Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[32] F. Glover and M. Laguna. General purpose heuristics for integer programming: Part i. *Journal of Heuristics*, 2:343–358, 1997.

[33] F. Glover and M. Laguna. General purpose heuristics for integer programming: Part ii. *Journal of Heuristics*, 3:161–179, 1997.

[34] J.-L. Goffin and J.-P. Vial. On the computation of weighted analytic centers and dual ellipsoids with the projective algorithm. *Mathematical Programming*, 60(1-3):81–92, 1993.

[35] J.-L. Goffin, A. Haurie, and J.-P. Vial. Decomposition and nondifferentiable optimization with the projective algorithm. *Management Science*, 38(2):284–302, 1992.

[36] J.-L. Goffin, A. Haurie, J.-P. Vial, and D. Zhu. Using central prices in the decomposition of linear programs. *European Journal of Operational Research*, 64(3):393–409, 1993.

[37] J.-L. Goffin, J. Gondzio, R. Sarkissian, and J.-P. Vial. Solving nonlinear multicommodity flows problems by the analytic center cutting plane method. *Mathematical Programming, Series B*, 76(1):131–154, 1997.

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

[39] F. Gzara and J.-L. Goffin. Exact solution of the centralized network design problem on directed graphs. *Networks*, 45(4):181–192, 2005.

[40] F. Hillier. Efficient heuristic procedures for integer linear programming with an interior. *Operations Research*, 17:600–637, 1969.

[41] J. Holland. Adaptation in natural and artificial systems. *Ann Arbor: University of Michigan Press*, 1975.

[42] D. Huisman, R. Jans, M. Peeters, and A. P. M. Wagelmans. Combining column generation and lagrangian relaxation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation, GERAD 25th Anniversary Series*, pages 247–270. Springer, Berlin, 2005.

[43] R. G. Jeroslow and T. H. C. Smith. Experimental results on Hillier's linear search. *Mathematical Programming*, 9(1):371–376, 1975.

[44] L. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6(4):366–422, 1960.

[45] M. Karamanov and G. Cornuéjols. Branching on general disjunctions. *Mathematical Programming*, pages 1–34, 2009.

[46] J. Kelley. The cutting plane method for solving convex programs. *Journal of the SIAM*, 8:703–712, 1960.

[47] A. Land and A. Doig. An automatic method for solving discrete programming problems. *Econometrika*, 28:497–520, 1960.

[48] H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):pp. 538–548, 1983.

[49] J. Linderoth and T. Ralphs. Mixed integer programming instances, 2007. (http://coral.ie.lehigh.edu/mip-instances/).

[50] J. Linderoth and M. Savelsbergh. A computational study of search strategies for mixed integer programming. *Informs Journal on Computing*, 11:173–187, 1999.

[51] A. Løkketangen and F. Glover. Solving zero-one mixed integer programming problems using tabu search. *European Journal of Operational Research*, 106:624–658, 1998.

[52] T. Magnanti and R. Wong. Accelerating Benders decomposition algorithmic enhancement and model selection criteria. *Operations Research*, 29:464–484, 1981.

[53] A. Mahajan and T. K. Ralphs. Experiments with branching using general disjunctions. In J. W. Chinneck, B. Kristjansson, and M. J. Saltzman, editors, *Operations Research and Cyber-Infrastructure*, volume 47 of *Operations Research/Computer Science Interfaces Series*, pages 101–118. Springer US, 2009.

[54] A. Mahajan and T. K. Ralphs. On the complexity of selecting disjunctions in integer programming. *SIAM Journal on Optimization*, 20(5):2181–2198, 2010.

[55] D. McDaniel and M. Devine. A modified Benders partitioning algorithm for mixed integer programming. *Management Science*, 24:312–319, 1977.

[56] P. Mervet. Fixed charge network flow problems: Applications and methods of solution. *Large Scale and Hirarchical Systems Workshop, Brussels*, 1977.

[57] M. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21 (6):1087–1092, 1953.

[58] J. Mitchell. Computational experience with an interior point cutting plane algorithm. *SIAM Journal on Optimization*, 10(4):1212–1227, 2000.

[59] J. Mitchell and B. Borchers. Solving real-world linear ordering problems using a primal-dual interior point cutting plane method. *Annals of Operational Research*, 62: 253–276, 1996.

[60] J. E. Mitchell and M. J. Todd. Solving combinatorial optimization problems using Karmarkars algorithm. *Mathematical Programming*, 56:245–284, 1992.

[61] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, 1999.

[62] J. H. Owen and S. Mehrotra. Experimental results on using general disjunctions in branch-and-bound for general-integer linear programs. *Computational Optimization and Applications*, 20:159–170, 2001.

[63] W. Rei, J.-F. Cordeau, M. Gendreau, and P. Soriano. Accelerating Benders decomposition by local branching. *Informs Journal on Computing*, 21(2), 2009.

[64] D. Ryan and B. Foster. An integer programming approach to scheduling. In A. Wren, editor, *computer scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling, North Holland, Amsterdam*, pages 169–280, 1981.

[65] G. K. Saharidis and M. G. Ierapetritou. Improving Benders decomposition using maximum feasible subsystem (mfs) cut generation strategy. *Computers & Chemical Engineering*, 34(8):1237–1245, 2010.

[66] G. K. D. Saharidis, M. Minoux, and M. G. Ierapetritou. Accelerating Benders method using covering cut bundle generation. *International Transactions in Operational Research*, 17(2):221–237, 2010.

[67] R. M. Saltzman and F. S. Hillier. A heuristic ceiling point algorithm for general integer linear programming. *Management Science*, 38(2):263–283, 1992.

[68] N. Shor. Cut-off method with space extension in convex programming problems. *Cybernetics*, 13(1):94–96, 1977.

[69] G. Sonnevend. New algorithms in convex programming based on a notion of "centre" (for systems of analytic inequalities) and on rational extrapolation. In K. Hoffman, J. Hiriat-Urruty, C. Lemarechal, and J. Zowe, editors, *Trends in Mathematical Optimization: Proceedings of the 4th French-German Conference on Optimization in Isree, West Germany, 1986*, volume 84, pages 311–327, 1988.

[70] P. Vaidya. A new algorithm for minimizing eonvex functions over eonvex sets. *Mathematical Programming*, 73:291–341, 1996.

[71] T. Van Roy. Cross decomposition algorithm for capacitated facility location. *Operations Research*, 34(1):145–163, 1986.

[72] F. Vanderbeck. On dantzig-wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, 48(1): 111–128, 2000.

[73] F. Vanderbeck and L. Wolsey. An exact algorithm for ip column generation. *Operations Research Letters*, 19:151–159, 1996.

[74] J. R. Vera and I. Derpich. Incorporating condition measures in the context of combinatorial optimization. *SIAM Journal on Optimization*, 16(4):965–985, 2006.

[75] P. Wentges. Accelerating Benders' decomposition for the capacitated facility location problem. *Mathematical Methods of Operations Research*, 44:267–290, 1996.

[76] Y. Ye. *Interior point algorithms theory and analysis.* John Wiley and Sons, 1997.

[77] D. Yudin and A. Nemirovski. Optimization methods adapting to the significant dimension of the problem. 38(4):513–524, 1977.